Task Modeling, Sequencing, and Allocation for In-Space Autonomous Assembly by Robotic Systems

Joshua N. Moser

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Mechanical Engineering

Erik Komendera, Chair
Alan T. Asbeck
John R. Cooper
Robert Hildebrand
Alfred L. Wicks

June 7, 2022
Blacksburg, Virginia

Keywords: Multi-Agent Autonomous Assembly, Mixed Integer Programming, Genetic Algorithms, Screw Theory, In-Space Assembly

Task Modeling, Sequencing, and Allocation for In-Space Autonomous Assembly by Robotic Systems

Joshua N. Moser

Abstract

As exploration in space increases through the use of larger telescopes, more sophisticated structures, and physical exploration, the use of autonomous robots will become instrumental to build and maintain the infrastructures required for this exploration. These systems must be autonomous to deal with the infeasibility of teleoperation due signal delay and task complexity. The reality of using robots in the real world without direct human input will require the autonomous systems to have the capability of responding to errors that occur in an assembly scenario on their own. As such, a system must be in place to allow for the sequencing and allocation of tasks to the robotic workforce autonomously, giving the ability to re-plan in real world stochastic environments. This work presents four contributions towards a system allowing for the autonomous sequencing and allocation of tasks for in-space assembly problems. The first contribution is the development of the Stochastic Assembly Problem Definition (SAPD) to articulate all of the features in an assembly problem that are applicable to the task sequencing and allocation. The second contribution is the formulation of a mixed integer program to solve for assembly schedules that are optimal or a quantifiable measurement from optimal. This contribution is expanded through the development of a genetic algorithm formulation to utilize the stochastic information present in the assembly problem. This formulation extends the state-of-the-art techniques in genetic algorithms to allow for the inclusion of new constraints required for the in-space assembly domain. The third contribution addresses how to estimate a robot's ability to complete a task if the robot

must be assigned to a task it was previously not expected to work on. This is accomplished through the development of four metrics and analyzed through the use of screw theory kinematics. The final contribution focuses on a set of metrics to guide the selection of a good scheduling method for different assembly situations.

The experiments in this work demonstrate how the developed theory can be utilized and shows the scheduling systems producing the best or close to the best schedules for assemblies. It also shows how the metrics used to quantify and estimate robot ability are applied. The theory developed in this work provides another step towards autonomous systems that are capable of assembling structures in-space without the need for human input.

Task Modeling, Sequencing, and Allocation for In-Space Autonomous Assembly by Robotic Systems

Joshua N. Moser

General Audience Abstract

As space exploration continues, autonomous robots are needed to allow for the necessary structures to be built in-space, on Mars, and on the Lunar surface. Since it is not possible to plan for every possible thing that could go wrong or break, the robots must be able to figure out how to build and repair structures without human input.

The work presented here develops a framework that allows the this in-space assembly problem to be framed in a way the robots can process. It then provides a method for generating assembly schedules that describe very good, if not the best way to complete the assembly quickly while still taking into account randomness that may be present. Additionally, this work develops a way to quantify and estimate how good robots will be at a task they have not attempted before. Finally, a set of considerations are proposed to aid in determining what scheduling method will work best for different assembly scenarios.

The experiments in this work demonstrate how the developed theory can be used and shows the scheduling systems producing the best or close to the best schedules for assemblies. It also shows how the methods used to define robot ability are applied. The work developed here provides another step towards autonomous systems that are capable of assembling structures in-space without the need for human input.

Acknowledgements

This work could not have been completed without the support of many people. First, I am grateful to my advisor Dr. Erik Komendera for his ideas, support, and insight over the course of this work. Thank you for the encouragement and insight that you provided. Additionally, I want to thank the rest of my PhD committee, Dr. Alan Asbeck, Dr. John Cooper, Dr. Robert Hildebrand, and Dr. Alfred Wicks. Your input and constructive feedback were invaluable in the completion of this work.

I want to thank Amy, Jacob, and Holly for their help with data collection, proofreading, and their willingness to discuss ideas. Your help was of great value throughout the process of completing this research. In addition, thank you to Mel, Nick, Liam, and Sam for their support and encouragement during presentations and demonstrations. I am grateful for all the work that Devin, Simon, Paolo, Dominic, Carl, and many other students put into helping with the preparation for experiments and demonstrations.

My thanks also goes to the support staff in the Mechanical Engineering department. Cathy, Ben, Jamie, Alida, Johnny, and Lance, who provided support to this research and the degree process. Your help with the administrative details and IT support were greatly appreciated.

I am also very grateful to Sandy, Russell, Breno, Brandon, Nicholas, Lucas, Megan, Pam, Kent, and many others for your support, encouragement, and prayer. The many conversations and times of study were a great encouragement to me and greatly enjoyed.

I can not say thank you enough to my brother Ryan and my parents Jim and Dena. You have supported me through all of my academic work, encouraging me each step of the way.

Most importantly, I am very grateful to the Lord. I would not be here today without Him. – Ephesians 2:4-10

Contents

List of Figures

List of Tables

Chapter 1

Background and Motivation

As the human race seeks to understand the universe and grow in our understanding of celestial objects, there is a need for building bigger and better telescopes, more capable of providing insight into what is around us [72, 102, 126]. In the pursuit of this understanding, exploration will also be undertaken with future goals that include sending humans off of the earth and onto the Moon and eventually onto Mars [128] for long term operations. A frontier of this space exploration has utilized robotic technology to manipulate objects and aid in the preparation of scientific experiments [38, 77]. As this front of in-space research and exploration expands, autonomous robots are going to be vital in the process. From building telescopes and space structures too large to launch from Earth [3, 21, 73] to building structures for human habitation on Mars or the Lunar surface [131], the use of robotic workforces will be instrumental in achieving the next step in this domain. A core capability of these systems will need to be autonomous assembly, robotic systems that are capable of constructing, maintaining, and repairing infrastructures [9, 102, 140].

1.1 In-Space Assembly

These in-space autonomous assembly problems will require a suite of autonomous subsystems to be successful. One such subsystem is the sensor network where cameras, encoders, and other sensors will be required to provide the necessary feedback for autonomy in the in-

space environment. In addition to the sensor network hardware, robots that are capable of completing the tasks present in this domain will also be required [56, 142] along with local robotic autonomy capable moving the robots around their environments. Modeling the kinematics and dynamics of the robot units as well as path planning capabilities for each unit will be necessary for the robots to maneuver and manipulate objects. These are active areas of research: developing methods to model the physical hardware and its degrees of freedom [8, 50, 56] alongside algorithms to plan the navigation of a robotic unit about a workspace [37, 116].

Task sequencing and allocation is also a vital requirement. In unstructured environments, robotic systems must be able to respond to new information or situations where tasks may not go as planned. The current state-of-the-art for in-space robot assembly operations are primarily done through teleoperation [118]. In the few cases where there is autonomy for assembly, it is limited and requires human supervision and interaction [21, 85]. This paradigm of requiring teleoperation or even human supervised autonomy is not viable for longer distance missions [121] due to high latency communications [80] nor for missions with increased complexity [118].

The task allocation problem is complex and becomes even more so when applied to a heterogeneous system of robots with a range of capabilities and constraints. While there are some existing techniques for task allocation, there is no clearly preferred architecture for in-space assembly [104]. Existing planners for space flight operations include systems such as ASPEN (Automated Scheduling and Planning Environment) which focuses on generating a sequence of low-level spacecraft commands from a set of high-level science and engineering goals [35]. Additional systems include CASPER (Continuous Activity Scheduling, Planning, Execution, and Replanning) which is a planning software system that seeks to repair existing space flight schedules [80] and EUROPA (Extensible Universal Remote Operations Planning

Architecture) which is a tool set for building and analyzing planners [16, 114]. GRAMMPS (Generalized Mobile Mission Planner System) is yet another planner used to solve a traveling salesman problem of sending robots to certain jobs [25].

Some research for autonomous assembly has been done for particular assembly scenarios such as Ikea furniture assembly [79], manufacturing cells [11], and construction [65] applications. However, the formulations utilized in these applications deal with simple assemblies, predetermined sequences, brute force computation, or combined task allocation and robot motion planning which do not scale well [65]. As a result, many of the assembly applications for in-space are not able to be fully autonomous and use manually defined task sequences [73]. A very recent work by [117] begins to address this with a proposed system that uses a state space that tracks the state of each module in a global context and utilizes a planning architecture that uses graph search algorithms to evaluate proposed assembly sequences.

In its current state, the task scheduling knowledge base is still missing important elements necessary for robotic assembly in the absence of human input. One such element is the modeling of robotic ability with chance of failure and how that impacts state transitions. For some scenarios, it is possible for a robot to seem appropriate from a proximity or speed standpoint. However, it is possible that the chance of failure for something like a precise motion is high even though the robot can get to the job faster. In this case, considering only the assignment time, the robot may seem like a good choice when in actuality, a slower robot would be a better choice from a chance of success standpoint. Since probability information could be important for some assembly scenarios, the problem definition should be able to articulate it.

In autonomous assembly in real world environments, it is likely something will not go as planned - whether it be due to an error in the robot autonomy, manufacturing defects in components, or even environmentally caused complications. This leads to a second element

that factors into the state representation formulation; the ability of the state formulation to accommodate state changes that occur as a result of repairs or undoing previously completed assembly steps [66]. It is foreseeable that these repair steps may not be as simple as completing the applicable assembly steps in reverse. An example of this would be if a truss component has a manufacturing defect that causes it to yield under an unexpectedly low load. To repair the structure, in addition to unfastening the damaged parts, additional tasks, such as supporting the structure, would be required that were not present in the original assembly. This is because original assembly process was such that the assembly could bear its own load, resting on the part that is now damaged and needs to be removed. A state transition formulation should be able to articulate this type of scenario for the in-space assembly problem where robots are not going to have human input.

An additional element not present in the current literature is a formulation for the narrowing of the set of valid robots for a task based on previously allocated tasks. For instance, to fix a strut component in place, the component must first be moved into position, aligning the ends with rest of the structure. The ends must then be fastened to the structure. One possible solution to this is to make the aligning process and the affixing process one task, with a requirement that the robot completing the task must be able to position the part and connect it into place. For a specialized robot, designed for this particular task, this framing would make sense. However, in a scenario such as in-space assembly where a team of robots are building or maintaining a structure, the number of task types will be greatly increased. Such a scenario lends itself to having multiple robots in a team equipped to do different types of tasks rather then a robot that can handle every type of task. Using the aligning and fixing tasks as an example, a grasping robot may be needed to complete the aligning and a robot with an attachment tool would be needed for the affixing task. In this framing, an optimal solution would require the grasping robot be assigned to both tasks (aligning and affixing)

since the part can not be moved after alignment until it is fixed in place. However, the robot with the connecting tool is only required for the second task. Scenarios such as this would require a special type of constraint that required the same robot for the same operation (grasping the part) between the two jobs. Designated here as the continuity constraint, the grasp operation in the two tasks must be fulfilled by the same robot even if multiple robots are capable of grasping. Such a constraint should be introduced into a problem formulation for autonomous assembly.

In addition to the above, more traditional elements must also be described. Precedence constraints dictating the ordering of assembly tasks and workspace travel considerations to account for robots travel between jobs are also necessary. The formulation must also be able to discretize the different tasks at hand into different job and robot contributions across those tasks. In its current state, the literature for this field does not present a framework or system capable of handling all of these considerations.

1.2 Thesis Statement

For an in-space assembly problem with constraints that describe the tasks and steps within the assembly and that articulate robot capability, there exist methods that discover optimal / near-optimal task sequences and allocations that enable heterogeneous teams of robots to complete the assembly.

1.3 Research Questions

The work of this dissertation will be focused on addressing the task sequencing and allocation assembly problem for in-space robotic autonomous assembly. It will first address the need of

a problem formulation capable of articulating elements applicable to this assembly problem domain. It will address the question of generating assembly schedules that are optimal or near optimal using the problem formulation. Since this dissertation is focused on the task sequencing and allocation side of the in-space autonomous assembly problem, it will not prescribe the optimal way to implement all of the features that would go into a deployable system such as autonomy architectures and the most computationally efficient formulation for kinematic and dynamic considerations, path planning, and sensing technologies. That being said, part of the task allocation problem includes the capability of robotic systems. In addressing how these may be modeled, a kinematic formulation for how to model robotic units will be discussed in the context of approximating autonomous robot ability. Finally, this dissertation will discuss considerations for choosing a scheduling system to utilize the problem formulation, providing some metrics to articulate the goodness of a scheduling system as it applies to the developed formulation and the in-space assembly problem. These contributions can be articulated as research questions to be addressed.

1.3.1 Research Question 1: Formulating a Definition

Research Question 1: How can the features and characteristics in the autonomous robotic assembly problem be formally described such that the information necessary for task sequencing and allocation can be defined and utilized in a schedule generation method?

This research question focuses on the development of a problem formulation that can articulate all of the elements in the in-space autonomous assembly problem needed for task

sequencing and allocation. The contribution of such a problem formulation will need to include descriptions of the different features present in the assembly problem, such as the robotic workforce and the components being assembled.

Additionally, this formulation will need to be able to describe the process necessary to complete the assembly. This will need to include a method for describing individual tasks to be completed and provide some way to discretize robot contributions for these tasks. As part of describing the assembly process, constraining features need to be accounted for that can articulate what robots can work on different parts as well as the ordering of different steps in the assembly.

Finally, assembly state articulation must be covered in the problem definition. A method needs to be developed that can represent the transition between states and provide a way to model the stochastic contributions that could impact successful transitions between states. It should also be able to accommodate the change that may occur in assembly state sequences due to the disassembly needed for repair that could arise in the in-space autonomous assembly problem.

1.3.2 Research Question 2: Solving for Optimal Solutions

Question 2: What is an optimal or near optimal way to sequence the tasks in an assembly problem given an objective function?

This research question focuses on the generation of an assembly schedule that sequences the tasks and allocates them to robots such that the assembly will be completed in an efficient

manner. It is desirable to be able to generate schedules that are as near to optimal as possible for a given object.

To answer this research question, a method should be developed to solve for the optimal solution given an objective or, if that is not feasible due to the complexity, seek to solve for a schedule solution that can give some measure or approximation of how close to optimal it may be. The resulting solution should be valid in that it does not violate any of the constraints in the assembly problem.

1.3.3 Research Question 3: Defining Robot Ability

Question 3: Can the robotic ability to complete tasks in an assembly problem be approximated if the robotic hardware can not experimentally be tested for the specific operations to generate processing time information?

In an autonomous in-space assembly scenario, there is the possibility a robot will be required to preform a task that it was not expected to complete (for example, a non-specialist robot may be required to complete a task if the specialist robot for that task was damaged unexpectedly). In such cases, it is not always possible to experimentally run through specific operations or tasks with a robot to determine if a robot can complete a task and, if it can, how long it will take to complete the task. To address this, an approximation of how long it might take the robot to complete a type of task is an important component of an autonomous task allocation system.

To answer this research question, a criteria of what defines robot ability in the context of

autonomous assembly should be developed. This will allow for methods to be defined that analytically approximate how well robots meet the criteria.

1.3.4 Research Question 4: Comparing Scheduling Methods

Question 4: Given the complexity present in the task sequencing and allocation autonomous assembly problem, what metrics can defined to determine what scheduling methods should be chosen for a given assembly scenario?

Scheduling and allocating problems can take on a large range of solution generation methods [93]. As such, it is beneficial to have a metric set to articulate how good a scheduling method is at solving the defined problem. Additionally, such a set should allow for some insight into what scheduling methods should be used under different conditions.

To answer this question, a set of metrics should be developed to describe how well the scheduling method solves the assembly problem and provide points to compare against other scheduling methods.

1.4 Dissertation Roadmap

To address these four research questions, this dissertation will take the following form: Chapter 2 will address **research question 1**. It will present a novel problem definition formulation that articulates the different elements present in the assembly. It will also develop a job shop scheduling problem (JSSP) formulation to describe the tasks and robot contributions.

Following this, a constraint set will be explained that articulates how the assembly must be completed. A Markov decision process (MDP) formulation will then be presented to describe the states in the assembly.

Chapter 3 will address **research question 2**. It will present a mixed integer programming formulation (MIP) that uses the problem definition formulation developed above and provide a set of constraints that can be solved. This will allow for the optimal solution to be found or some measure of optimality based on the gap found using branch and bound.

The following chapter, Chapter 4, also addresses **research question 2**. It provides a genetic algorithm formulation to solve the scheduling problem using the developed problem definition. This formulation will allow for better use of the stochastic information in the problem formulation and has the potential to provide solutions faster (at the cost of optimality) than the MIP for larger problems.

Chapter 5 will address **research question 3** by presenting a set of criteria to classify and implement robotic ability using a kinematic formulation to approximate the ability of a robot to complete a task.

The last chapter of developed content, Chapter 6, will answer **research question 4** by presenting a set of criteria that can be used to describe how good a scheduling method is at solving an assembly problem presented in the developed problem formulation. It will also discuss how these criteria can be used to choose a scheduling method for different applications where the autonomous task allocation problem is present.

Chapter 7, the final chapter, will walk through areas of future work that can extend the different areas of research presented here and summarize how the thesis statement was addressed through answering the research questions in this dissertation.

Chapter 2

Stochastic Assembly Problem Definition

As discussed in Chapter 1, a requirement to generating efficient solutions to the task sequencing and allocation problem is a problem definition that successfully articulates the elements involved in the assembly. First, there are the physical features present in the problem. One such feature set covers the components that need to be assembled. Pertinent assembly information about the components may include: where the component starts out in an assembly, where it needs to be placed, how much it weighs, if it is damaged, etc. A problem formulation must provide a framework for this information to be encoded. Tangential to the component feature set is the connection method set, or joint information. In the context of an assembly, this feature set will include information such as: what type of connection, information about if the connection is completed or if it is damaged, and what components are being joined in the connection. Thirdly, the assembly space is another physical feature set in an assembly problem. Location information directly impacts the assembly, requiring some way to note spaces that are important in the assembly. Finally, in addition to the information about what is being assembled, the way the assembly comes together, and where it will take place, the operators completing the assembly must be defined. In the context of robotic assembly, this must define the robotic agents. This includes information such as a robot's ability to complete the tasks in the assembly, where the robot is located, how much energy it may

have left, etc. A general problem formulation for this type of assembly should provide a framework that is capable of articulating these features.

Just knowing what physical components are present in an assembly is not sufficient to define an assembly problem. As implied in the name, an assembly requires a pair or more of elements to be arranged in a certain manner differently then how they started, often to create a new element. In many assemblies, the way this is done is constrained. One such constraint often present is a limitation or requirement in order of how tasks are completed, requiring some tasks to occur before or after others. Additionally, limitations may exist for which robot can work on certain steps, either due to capability limitations or by design. Modeling how an assembly can take place must also be included in the problem definition if a system is going to autonomously sequence and allocate the tasks in the assembly.

Knowing the physical features of an assembly and all the constraints regarding how an assembly must take place is still not enough information to fully articulate the sequencing and allocation problem for some real world situations in a manner for an optimal schedule to be found. The remaining information is the state of an assembly. The current state of an assembly and modeling how the states may change should also be part of a problem description. In a real world situation, there is the potential that something will not go as planned. A fully autonomous system must have the information at hand describing the current state, what paths there are to change it, and how good or likely those paths are to succeed.

If a problem formulation can articulate all of the features described above it can provide the necessary information to articulate an assembly problem that an autonomous robotic system may encounter. In this chapter, the Stochastic Assembly Problem Definition (SAPD) has been developed for this purpose. It will provide the framework necessary to fully describe the information needed for this problem type in a format that can easily be utilized by schedule

generating methods.

2.1 Research Gap

Several different planners have been utilized in space. One such framework is ASPEN, an object-oriented system that allows users to define constraints applicable to their domain, designed to interface with AI scheduling algorithms [55]. GRAMMPS is another system which uses a grammar to increment robotics, jobs, precedence, and moving a robot to a goal location. It can note two robots with an AND or OR operator. The planning component takes a mission statement embedded in the grammar and seeks to solve the traveling salesman problem (TSP), using randomized searches as necessary [25]. CASPER [80] is a system that focuses on repairing existing schedules, correcting errors using an integrative repair approach. Inheriting from ASPEN, it takes high level goals and seeks to generate a plan utilizing a finite-enumeration state set, fuel information, task hierarchy, and action time constraints (how long something takes).

A common description framework present for planning tasks for robotic systems not specifically designed for space is the Planning Domain Definition Language (PDDL) [5, 14, 53, 54]. PDDL uses a set of object types to provide a framework for describing elements of a planning problem such as actions that need to be taken during the planned process [6]. There have been different variants, adding elements such as time duration considerations [52] and a framework to interface with constraint engines [16]. Such frameworks seek to standardise automated planning languages [26] and have been utilized in commonplace robotic systems such as the Robot Operating System (ROS) [28]. However, they do not contain the formal structure or completeness necessary for a rigorous standalone problem formulation for the autonomous robotic assembly problem.

Other assembly description frameworks such as SysML [68, 69] seek to provide a taxonomy for assembly actions for robots. In this framework, assembly actions seen in manufacturing scenarios such as transport and insert are classified. Constraints such as speed and degree of freedom are also discussed. [27] focuses on a list of robot capabilities, linking processing times to different abilities for different sets of robots.

Still other frameworks, such as PPRS [111] provide a skill definition framework to distinguish different types of tasks such as basic tasks, those with well defined durations, and compound tasks, those requiring multiple sets of skills such as grasping and movement. In the context of assembly line design, assembly sequences are based on semantic descriptors of a product and require a taxonomy of assembly operations [24, 86]. These taxonomies have entry locations for elements such as task parts and a task database. [143] focuses on defining a task in planning scenarios, providing a framework for actions it will require, the goals of the task, any sub-tasks, etc.

Frameworks such as the one in [99] use Petri Net based approaches on the autonomous control side of task planning to model the states and transitions with a focus on the agent itself. [31] focuses on the state transition side of planning using MDPs and Partially Observable MDPs as a generic state modeling system. Still other formulations for assembly focus primarily on a very fine resolution level, such as describing how components must go together and describing valid orientations and positions [97].

These frameworks consider one or a few of the features present in the autonomous assembly problem. However, a complete formulation system, capable of formally defining each element and constraint type, feeding into a state representation that can model the ability of replanning or job insertion is not present in the literature. The following sections will discuss the problem formulation developed in this work to fill this gap in the literature.

2.2 Formulation

To provide a problem formulation capable of adequately defining the autonomous robotic assembly problem for the purpose of task sequencing and allocation, the following SAPD has been develop. This formulation, published in [101], takes the form of three main groups: *Elements*, the physical and functional characteristics; *Constraints*, to articulate a valid assembly sequence; and *State Representation*, a flexible mathematical model to represent the overall state of the assembly in the context of a stochastic paradigm.

2.2.1 Elements

The *Elements* portion of the problem formulation describes the "physical" portions of assembly agents, tasks, and environment features present in the assembly problem. This is done through four different classes: *Poses*, the important locations and orientations in the assembly space, *Components*, the structural parts in the assembly, *Joints*, the structural connection information, and *Robots*, the autonomous operators forming the workforce in the assembly problem. Each of these types will contain a property category that will be used as a bookkeeping / information storage area containing the information that *does not* contribute to the current state of the assembly. This can include information such as the element's name, type, starting location at the beginning of the assembly, etc. The classes that contain the information that *do* affect the state of the assembly will have a second category, states. These stateful information entries will include information such as the current location of the element at a specific time in the assembly, if the element is in position or not, if it is damaged, etc. The following sections expand on the formulation structure for each of these element classes.

Poses: To describe poses of interest in the workspace a *pose* class will be defined. This class is primarily used to describe locations of interest through the assembly environment though it can be used to highlight other types of poses as needed. Denoted by the set $\boldsymbol{W} = \{W_1, \ldots, W_i, \ldots W_{|\boldsymbol{W}|}\}$, each pose can be thought of as a description of a specific location, position, or position and orientation. Since these designations do not typically change as the assembly progresses, this class will be stateless in most cases. Each pose will contain at least two property features: type and location (containing both position and orientation when applicable). The type feature allows for different categories of poses to be distinguished between and in most cases, at least a reference type will be present to note the different locations. The location feature provides the coordinate of the poses in the workspace. It can also be a location relative to a different frame of reference. An example of this would be referencing a pose on a component. The position would be with respect to the component frame so that it did not change every time the component moved and unnecessarily increase the number of states required to describe the system (the state definition of the assembly will be discussed in more detail later in this chapter). The overall formulation does not restrict the type of coordinate frame (Cartesian, cylindrical, etc.) or specifically require orientation information to be included, leaving it open to what best fits the assembly problem environment and application. Additional property information can be added if required in the context of the assembly.

Components: The next class is the *component* class. This class, represented in the set $\boldsymbol{C} = \{C_1, \ldots, C_i, \ldots C_{|\boldsymbol{C}|}\}$, is any physical part that is included in the autonomous assembly problem and is not an autonomous operator. This set contains all of the elements that will need to be manipulated for the assembly. Like the pose class, each C_i has a set of properties describing the type of component and other information that may pertain to the assembly. The components directly affect the state of the assembly. Therefore, each

component is stateful and will contain a set of state information features. There are generally three different state features present in assemblies: in position / not in position, broken / not broken, and current location. The first state, in position / not in position, reflects if the component is in the correct installation location. If it is not, additional actions will be required to move it into position. The broken / not broken state describes if a component is going to need to be replaced or if it can remain. Noting both of these states together allows for the description of a condition where a component might be in place but installed wrong or the condition where a component has been broken after installation and needs to be removed. This is an important distinction for cases where replanning may be required. Either case of being in the broken state will require a task or task set to be inserted into the assembly sequence to correct the problematic component. Finally, the current location state provides the location information as the component transitions around the environment. Similar to the pose class, the components class contains a property feature set for each component which will contain information such as component type, locations, and weight (if applicable). The location property feature can contain poses that are important to the component. Two primary examples are the start location at the beginning of the assembly and the goal location. An additional example includes poses on the component necessary for joining (this ties into the second use case of the pose class described in the previous section). As previously noted, including these locations in the component and defined with respect to the component, their positions and orientations can be with respect to the component's location rather than the global reference frame. This removes the need for them to be in the state feature category and thereby reduces complexity in the overall assembly state representation.

Joints: A *joint*, represented in the set $\boldsymbol{K} = \{K_1, \ \ldots, \ K_i, \ \ldots \ K_{|\boldsymbol{K}|}\}$, can be thought of as an element of the assembly problem that is physical but is not in the form of a component

or a location. It represents the connection between components in an assembly - taking the form of welds, bolt joints, etc. If needed, and with a slight abuse of some terminology, this can be used to describe painting or spreading a sealant on components as well. Similar to the component type, this element contains states and properties in its definition. For a joint, joined / not joined or completion percent of being joined are the two most common state features. If there is a process to completing a joining method, the percentage state representation may be necessary. Alternatively, if the connection is something like a snap connection, having a simple joined / not joined state representation can be sufficient to describe the state. Type, locations, and component list are all examples of the property features that may be included in the joint definition.

Robots: The *robot* element class is the final of the four and represents the autonomous operators used to complete the assembly project. A given robot, represented in the set $\boldsymbol{R} = \{R_1, \ \ldots, \ R_i, \ \ldots \ R_{|\boldsymbol{R}|}\}$, will have the following state features: idle / busy, current location, current task, and energy level (if applicable). The idle / busy state is used to denote if a robot is available for assignment. The current location, as implied, gives the updating location of the robot and current task represents what the robot is currently working on. This state does not remove the need for the idle / busy state since there are many applications where a robot may not be tasked but is still not available for a task allocation. Energy level, if the information is applicable, can be thought of as a projection of how much longer the robot can operate before it needs to be retired for charging or replacement. In addition to the states, each robot will generally have five property features: robot type, locations, mobility, workspace, and abilities. Mobility and workspace, when present, provide enough information to articulate the reach of a robot (the local workspace) without moving the base, and if a robot is mobile, that is, if the origin of the local workspace can be moved to a new location in the environment. The combination of these two properties allows for a schedule

generator to determine if a task is reachable by a specific robot.

To define the abilities feature, a set of operation types in the project needs to be defined. These are based on the types of components, joints, and robots in the assembly project and represents the discretized contribution of a robot in a given task. These operations, defined as the set $\boldsymbol{O} = \{O_1, \ \ldots, \ O_i, \ \ldots \ O_{|\boldsymbol{O}|}\}$, will be described in more depth in Section 2.2.2. For now, it is enough to say that each robot will have an entry for each $O_i \in \boldsymbol{O}$ in its abilities property describing its capability or limitations in completing that operation. In its simplest form, this can be thought of as a processing time, describing how long the robot will take to complete the operation. In higher fidelity formulations this entry will include stochastic information in the form of distributions. The processing time, approximated as a normal distribution, will describe the variation present in completing the specific operation by the robot. This can include variation introduced from a need to adjust the robot's grip on a component or correcting for an overshot location. Additionally, probabilistic information for minor and major failures can also be included at this point. Minor failures, defined as failures in completing an operation that do not break the component or portions of the assembly, can be separated out of the processing time and modeled as a separate normal distribution paired with a geometric distribution to model the number of times the robot might need to attempt aligning a component and how long each occurrence may take. The chance of a major failure, one that damages other elements in the assembly triggering a need for new tasks to be inserted to undo, replace, or repair the damaged components can also be modeled with a Bernoulli distribution. All of these distributions can be replaced to fit the characteristics for the specific robots and scenarios in a given assembly without changing the architecture of the SAPD.

2.2.2 Constraints

The constraints portion of the SAPD frames the requirements describing the criteria for a valid assembly. This will includes a job shop type formulation to discretize and articulate the steps needed to complete an assembly, precedence constraints to ensure tasks are completed in the required order, continuity constraints and machine validity constraints to ensure that the correct machines are used on tasks, and finally, distance constraints to encapsulate the impact of robots traveling between tasks in the assembly.

Job Shop Scheduling Problem Formulation: Variants of the job shop scheduling problem (JSSP) formulation have been used throughout literature as a way to describe a project in terms of the tasks that need to be completed, represented as jobs, and the individual machine contributions in each of those tasks, represented by operations [33, 151]. The in-space robotic assembly problem falls into the category of flexible job shop problems (FJSP). This is an extension of the NP-hard classical job shop scheduling problem noting that a task can be processed by any machine from a list of machines. This category of the JSSP has two subproblems in it: the routing subproblem to select a suitable machine to process the task and the scheduling subproblem to sequence the tasks [32]. The problem classification is further extended with the additional complexity of there being multiple ways that jobs could be completed. This leads to the categorization of the problem being a flexible job shop scheduling problem with process plan flexibility (FJSP-PPF) [150] also referred to as the FJSP with integrated process planning (FJSP-IPP) [88] or alternative process plans (FJSP-APP)[113]. In addition to the complicating factors, this particular formulation has yet another complicating element where some pairs of jobs require the same robot unit to work on this. These continuity considerations add a new complexity. The following sections will develop these different elements further.

In this paradigm of the JSSP, the jobs, represented by the set $\boldsymbol{J} = \{J_1, \ldots, J_i, \ldots J_{|\boldsymbol{J}|}\}$, make up the tasks that must to be assigned and processed to complete an assembly problem. Each type of job will have a set of operations ($\boldsymbol{O_j}$) as sub-elements representing different robot contributions in completing a job. Therefore, if a job can be completed by one robot alone, it will contain one operation. However, if a method of completing a job requires two robots, that will be represented by two different operations. In some cases, there may be more than one way to complete a job. This is represented by a set of process plans ($\boldsymbol{P_j}$) where each process plan is a set of operations describing how job j can be completed.

Precedence: Many assemblies require specific task sequences to successfully reach the completed state. This is reflected in scenarios such as a component can not be connected until it has been moved into position. To represent this constraint set, a Directed Acyclic Graph (DAG), $G_p(V_p, \alpha_p)$, is used to embed the precedence constraints. Each vertex, V_p, represents a job in the assembly project and each arc, α_p, represents a precedence constraint, thereby encoding the project precedence constraints in the structure of the graph. Representing precedence in this way allows for ordering constraints to be described only in terms of the jobs that must immediately proceed or follow it to provide a sequence framework that the overall assembly must follow to reach a valid completion state.

Continuity: In an assembly project, there may be occasions where a robot needs to continue the same operation between two jobs. An example of this is illustrated by a robot tasked with aligning a part before it is fixed into position. The same robot must also be tasked with holding that part during the actual affixing job since the use of a different robot would result in the loss of alignment. By using the precedence DAG to require the two jobs to come in direct sequence and the continuity constraint requiring the robot to be constant across the two jobs, the resulting assignment will reflect the requirement of using the same

robot for the two tasks without being assigned something in-between. For this formulation, the continuity constraint, represented by the set $\boldsymbol{H} = \{(\alpha_p, O)\}$, is framed as arcs from the precedence DAG and operation type parings. The arcs describe which jobs these constraints apply to and the operation type represents which operations needs to be assigned to the same robot across the arcs.

Valid Robot Section: In an assembly problem, there are instances in which a specific robot must be used for a job even if other robots are technically capable of completing it. This constraint can be used to force certain robots to work on a specific job. It can also be used to reduce the number of job types. For example, if the robots each have a specific job reflecting how they are stored at the end of an assembly, only one storage job type needs to be defined. This constraint can then be used to require individual robots to complete the storage job that is applicable to them. To formulate this constraint, a set of pairs, $\boldsymbol{V} = \{(J, R)\}$, is defined that represents what robots can complete what jobs.

Distance: Finally, spatial information, part of the *environment* consideration, must be included for many assembly problems. This consideration is modeled using a fully connected graph (FCG), $G_d(V_d, E_d)$. There are two different ways that this constraint can be implemented. The vertices, V_d, can either represent specific jobs or the locations of poses in the assembly space. In both cases the edges, E_d, represent the distances between the features modeled by the vertices. Depending on what other autonomous capabilities are present or what additional information is known about the environment, these edge values can represent direct pose to pose distance, such as a Cartesian distance, or they can represent the distance of a path that must be taken to move between the vertices. In either case, the functionality of the FCG remains the same and provides information to the schedule generator allowing it to determine the time commitment and requirement of moving between specific jobs or

locations for a given robot.

2.2.3 State Representation

The definitions above provide a description method for each element in the assembly project and their individual states, when applicable. The aggregated states of all these elements form the basis for a description of the overall state of the assembly. For the purpose of task assignment, it is necessary to model how the state of an assembly will change for a particular robot to task (job / operation) assignment. As noted earlier, a realistic autonomous assembly problem will often contain stochastic elements that factor into the state transition of the assembly. To accommodate this, the SAPD models the autonomous assembly as a MDP, represented as the tuple $(\boldsymbol{S}, \boldsymbol{A}, P(s'|s,a), R(s',s))$, containing the state set, action set, transition probability, and transition reward respectively.

State Space: In the MDP formulation, $\boldsymbol{S}$ is the set of all possible states the assembly can take, $\boldsymbol{S} = \{S_c, S_J, S_R\}$. Each unique state, $s \in \boldsymbol{S}$, is a different combination of the stateful elements present in the assembly. Many of these are naturally discretized as in the case of a component being in position or not. Other states, such as location and percent completion, must be approximated by a discretized unit. The level of discretization will be a function of the schedule generator's sensitivity to state space size. Regardless of the level of discretization, the following formulation is still valid.

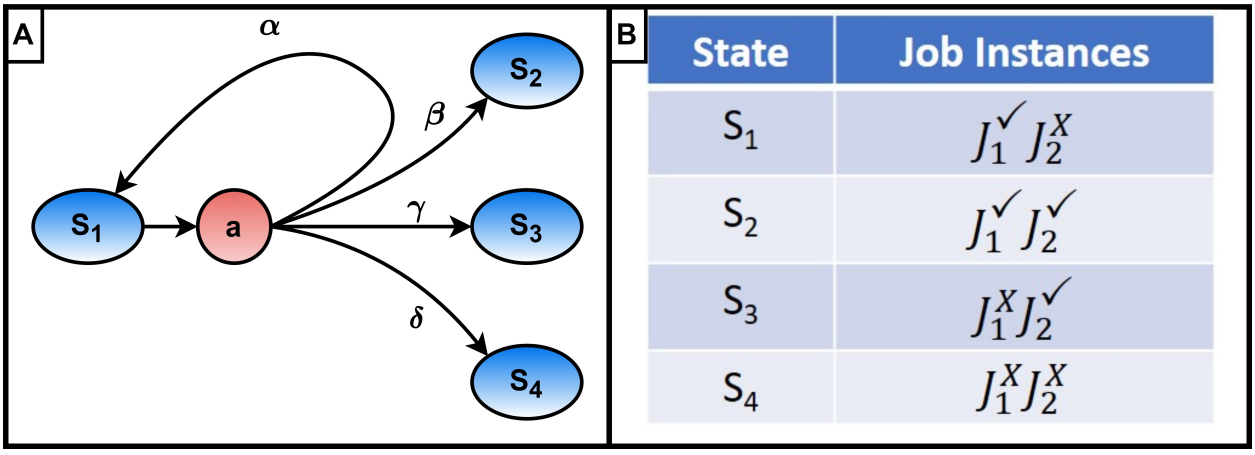
Action Space: The action space, $\boldsymbol{A}$, is the space representing all of the possible actions, a, in the assembly problem. These actions take the form of assigning different robots to different job / operation pairs: $a : r \rightarrow (j,p,o)$ where $r \subseteq \boldsymbol{R},\ j \subseteq \boldsymbol{J},\ p \in \boldsymbol{P_j},\ o \subseteq \boldsymbol{O_{jp}}$. In an autonomous assembly scenario, the schedule generator will assign these actions during

the task allocation process.

Transition Probability: The transition probabilities, $P(s'|s,a)$, can be thought of as a measurement of how likely a certain state transition is $(s \rightarrow s')$ given an action assignment (a). These transition probabilities are a key feature of this problem formulation, allowing it to encapsulate much of the stochastic nature present in the assembly problem. These probabilities are sourced from the abilities in the SAPD formulation for the robot class described earlier. The model functionality is demonstrated in the simple MDP graph in Figure 2.1 Panel A where a simple assembly requires a block to be moved into position and connected to the ground. Letting J_1 represent moving the block and J_2 connecting it in place, state s_1 represents the state where the block has been moved into position but has yet to be connected. The other three states are shown in Figure 2.1 Panel B and represent the block fastened in the correct location (s_2), fastened in the incorrect location (s_3), and finally, not fastened and not in the correct location (s_4). Based on the action assignment a, a robot $(R1)$ is assigned to connect the block into position. If there is a 25% chance of the robot failing to complete the connection task ($P_{R_1}(J_2^{\text{✗}}) = 0.25$), a 50% chance that the failure will dislodge the block from its correct location $P_{R_1}(J_1^{\text{✗}}|J_2^{\text{✗}}) = 0.50$), and a 0.1% chance a successful connection will dislodge the block ($P_{R_1}(J_1^{\text{✗}}|J_2^{\text{✓}}) = 0.001$), the transition probabilities of ending up in the four different states are:

$$\begin{aligned} P_a(\alpha) &= P_{R_1}(J_1^{\text{✓}}|J_2^{\text{✗}})P_{R_1}(J_2^{\text{✗}}) = 0.125 \\ P_a(\beta) &= P_{R_1}(J_1^{\text{✓}}|J_2^{\text{✓}})P_{R_1}(J_2^{\text{✓}}) = 0.7425 \\ P_a(\gamma) &= P_{R_1}(J_1^{\text{✗}}|J_2^{\text{✓}})P_{R_1}(J_2^{\text{✓}}) = 0.0075 \\ P_a(\delta) &= P_{R_1}(J_1^{\text{✗}}|J_2^{\text{✗}})P_{R_1}(J_2^{\text{✗}}) = 0.125 \end{aligned} \tag{2.1}$$

By structuring the state representation with this MDP structure, the possibility of *failures*,



State	Job Instances
S_1	$J_1^{\checkmark} J_2^{X}$
S_2	$J_1^{\checkmark} J_2^{\checkmark}$
S_3	$J_1^{X} J_2^{\checkmark}$
S_4	$J_1^{X} J_2^{X}$

Figure 2.1: Panel A: Example Markov decision process state transition graph, Panel B: States in example Markov decision process.

restarts, and a need for *backtracking* to repair in an assembly project can be quantified and taken into account by the autonomy. This form also allows for impossible paths to be blocked by asserting zero probability for the impossible transitions. In this general form, *job insertions* are not an issue since they would simply represent a different action option at states. Unknown states are not a concern since this type of formulation describes **every possible state** by design. This is an important element when it comes to replanning since the required transitions for replanning are *already embedded* in this formulation. It should be acknowledged here that while this is possible in the theoretical case, in practice, extremely large state spaces are not practical for many schedule generator methods to handle. Therefore, some schedule generation formulations will use a reduced set of states in the MDP model to approximate the entire assembly. However, the formulation itself is capable of scaling as required for the specific assembly problem and schedule generation method being utilized.

Transition Reward: The final element of the MDP formulation is the state transition reward, $R_a(s', s)$. This represents how beneficial a state transition $(s \rightarrow s')$ is with respect

to completing the assembly. This might take the form of the inverse cost of transition, a value rewarding specific job completions, a measure of how close the new state is to project completion, or focusing on elements such as penalizing the entering of broken states (states where something in the assembly is broken). The general formulation does not constrain how this reward is generated, leaving a framework in place to be utilized by the different schedule generation methods. Some examples of the reward might be a time cost, or some penalty value utilized in a schedule generator's solver.

2.3 Summary of Contribution

The SAPD presented in this chapter addresses the first research question inquiring how the task sequencing and allocation problem for autonomous robotic assembly can be modeled. First, the different types of features are categorized as physical elements, constraints, and state representation. A formulation was presented for the physical elements that provided a framework capable of articulating both stateful and non-stateful characteristics. Next, a constraint set was developed, utilizing existing concepts from different domains such as job shop scheduling and developing the constraint type of continuity constraints, a requirement for collaborative assembly work by a heterogeneous robotic systems.

Additionally, a new way to think about the state representation for this type of problem was developed using a MDP where the transition probabilities structure is informed by the earlier formulation components and can easily be expanded as more statistical information is known for the assembly problem. The formulation naturally incorporates the difficulties that would come from replanning such as state insertion. Simultaneously, it was demonstrated how this formulation can also be used to articulate non-possible state transitions through the probabilities.

This work contributes to the literature field by providing a formulation that is scalable and adjustable to accommodate different scheduling methods that may be developed to take advantage of different features present in a range of assembly scenarios. Two such scheduling methods have been developed in this dissertation and will be discussed in Chapters 3 & 4 respectively.

Chapter 3

Task Assignment: Solving for the Optimal Solution

Now that a framework has been developed to describe the in space autonomous assembly problem, different solution methodologies can be developed and implemented to produce assignment schedules to describe task sequences and robot to task allocations that lead to a valid assembled state. To address the second research question, a mixed integer programming solution (MIP) was chosen for its ability to solve computationally complex allocation problems such as the traveling salesman problem [19, 75, 112, 145], the vehicle routing problem [29, 58, 138], and other scheduling problems such as medical staff scheduling [7, 107] while providing some measure of how close to optimality a solution is through the branch and bound solution framework. Using linear relaxations of the MIP formulation and by varying the MIP problem, a search tree can be formed and solutions can be found that satisfy the original MIP. Once a feasible solution is found, that solution can be thought of as the upper bound of a range within which the optimal solution must fall (assuming the problem is a minimization problem). The lower bound is defined as the minimum optimal objective value of the sub-problems in the solution tree. The difference of these two bounds provides a gap within which the optimal solution must be located. Once this gap converges to zero it is known that the optimal solution has been found.

3.1 Research Gap

Current mixed integer programming formulations for robot task assignment include considerations for distance [23], job sequence considerations [57, 108], and cross robot dependencies [82]. Other applications for robot task allocation modeled with a MIP include human and robot collaboration (HRC) [91] and dual-arm task assignment [130]. Still others attempt to solve the scheduling problem with additional considerations such as building ad hoc networks [51]. The formulation presented below, published in [101], expands the concepts in these formulation to include considerations for robotic task assignment by framing the constraint set to account for sequence dependent operation requirements and local workspace considerations with respect to the robots.

3.2 Formulation

The developed MIP formulation can be thought of in four main sections: *parameters*, *variables*, *objective function*, and *constraints*. The *parameters* articulate elements in the problem that will remain constant throughout the process. The *variables* describe the binary and continuous elements being modified by the MIP solver as it seeks, in this case, to minimize the *objective function* within the limitations of what makes a valid assembly sequence and task allocation described by the *constraints*. First, a mathematical representations of the SAPD elements must be mapped for use in the MIP formulation.

3.2.1 Sets

The sets of robots, jobs, process plans, and operations sets from the SAPD are all used directly in the MIP formulation. Additionally, a set of arcs from the precedence DAG must

be defined to include the precedence information into the MIP formulation domain. Using these arcs, the continuity constraint information is also included. In the MIP formulation, all of these sets are defined as:

R	Set of all robots
J	Set of all jobs
P_j	Set of all process plans for job j
O_{jp}	Set of all operations for a given process plan P_j
$\mathcal{A}$	Set of all precedence arcs
$\mathcal{O}_a$	Set of all operations that require continuity for a given $a \in \mathcal{A}$

3.2.2 Parameters

In addition to the sets, several parameters (values that will not change within the model during solving) must be defined. The first is the processing time, a representative value pulled from the robot ability information. This value is used by the scheduler to model how long it will take a given robot to complete a certain operation. In addition to the process time, a time for traversing distance must be defined. This value, noted as the setup time since it is the time a given robot will take to move from one job to another, is derived from the distance between two jobs, pulled from the FCG in the SAPD and the given robot's ability to move around the environment. Additionally, parameters describing if a robot is mobile, if a job begins within a robot's local workspace, and if a specific job can be allocated to a specific robot are also defined from the SAPD. Finally, a very large constant number must be established for use in the MIP constraint definition to allow for certain constraints to be "turned off" under certain conditions. This will be explained further in Section 3.2.5.

The definition of these parameters take the form:

T_{jpor}	The processing time of operation o in the process plan p for job j by robot r
$S_{jj'r}$	Setup time of robot r moving from job j to job j'
m_r	1 if robot r is mobile and 0 if it is not
g_{jr}	1 if job j is within the local workspace (grasping range) of robot r and 0 if it is not
v_{jr}	1 if robot r can work on job j and a 0 if it can not
L	A very large number used to help toggle constraint equations

3.2.3 Variables

Binary: The binary variables represent decisions and can be thought of as a 1 or 0 switch, a 1 if the decision was a yes and a 0 if the decision was a no. These decisions include if a robot is assigned to a specific job, if a specific process plan is chosen for a given job, and what operation a robot is assigned to for a given job, process plan, and operation configuration. They also include what job a robot starts working on, what jobs it transitions between, and what job it ends on. The definition for each of these binary variables take the form:

x_{jr}	1 if job j was assigned to robot r, 0 otherwise
x_{jp}	1 if process plan p is chosen for job j, 0 otherwise
x_{jpor}	1 if robot r is assigned to operation o in process plan p for job j, 0 otherwise
x_{jr}^{start}	1 if robot r starts project with job j, 0 otherwise
$x_{jj'r}$	1 if robot r moves from job j to j', 0 otherwise
x_{jr}^{end}	1 if job j is the last job that robot r works on in the project, 0 otherwise

Continuous: In addition to the binary variables, there is a set of continuous variables used to note important time quantities that the model must set. These include when a job is started, when it is completed, when a specific operation is completed within a job, and when a robot completes its work on a job. Additionally, the times when a robot starts traveling between two jobs and finishes traveling between two jobs are also modeled. Finally, the overall project duration (the project makespan in job shop terms) is defined. In this formulation, these take the form:

s_j	The start time of job j
c_j	The completion time of job j
c_{jpo}	The completion time of operation o in process plan p for job j
c_{jr}	The time robot r completes job j
$s_{jj'r}$	The start time for robot r moving from job j to job j' for $j \neq j'$
$c_{jj'r}$	The completion time for robot r moving from job j to job j' for $j \neq j'$
$c_{\max}$	The upper bound on completion time (the project makespan)

3.2.4 Objective

For this formulation the objective is to minimize the overall time it takes to complete the assembly project. This will drive the model to minimize the amount of time a robot is idle and it will encourage the solver to choose robot assignments that pick the best robot for a given job / operation while taking into account how long it will take the robot to move to the job it has been assigned. The objective function is defined as:

$$\min \ c_{\max} \tag{3.1}$$

3.2.5 Constraints

To enforce all of the constraints present in the problem formulation a series of linear constraints must be defined. These can be broken into five different groups: assignment constraints, continuity constraints, robot flow constraints, mobility and workspace constraints, and finally, timing constraints.

Assignment Constraints: In the first set of constraints, the fact that a job can only be performed one way across the whole assembly is enforced. This means only one process plan can be used per job in the assembly (3.2). When a robot is assigned to a job it must also be assigned to one and only one operation in that job (3.3) & (3.4), which must be included in the list of operations in the process plan the scheduler selected (3.5) & (3.6). If the robot is not assigned to a job it can not work on any operations in that job (3.7). These principles are enforced by the following constraints:

$$\sum_{p \in P_j} x_{jp} = 1 \qquad \forall j \in J \tag{3.2}$$

$$\sum_{p \in P_j} \sum_{o \in O_{jp}} x_{jpor} \leq 1 \qquad \forall j \in J, r \in R \tag{3.3}$$

$$x_{jp} = \sum_{r \in R} x_{jpor} \qquad \forall j \in J, p \in P_j, o \in O_{jp} \tag{3.4}$$

$$x_{jpor} \leq x_{jp} \qquad \forall j \in J, p \in P_j, o \in O_{jp}, r \in R \tag{3.5}$$

$$x_{jpor} \leq x_{jr} \qquad \forall j \in J, p \in P_j, o \in O_jp, r \in R \tag{3.6}$$

$$x_{jr} = \sum_{p \in P_j} \sum_{o \in O_{jp}} x_{jpor} \qquad \forall j \in J, r \in R \tag{3.7}$$

Continuity Constraint: If there is a continuity requirement, that is, if the same robot must work the same operation between two jobs, that is enforced with the constraints:

$$x_{j'p'or} \leq x_{jpor} \qquad \forall p' \in P_{j'}, p \in P_j, o \in \mathcal{O}_{(j',j)}, (j',j) \in \mathcal{A}, r \in R \tag{3.8}$$

$$x_{jj'r} \geq x_{jr} \qquad \forall (j',j) \in \mathcal{A}, r \in R \tag{3.9}$$

Robot Workflow Constraints: In addition to the assignment considerations above, the manner in which robots work on jobs must also be enforced. In a robot's workflow, there must only be one start and one end node (3.10) & (3.11). A robot must only start and stop working on a given job once in its workflow before moving on to the next job (3.12) & (3.13), and it must not loop back to the same job after completing it (3.14):

$$\sum_{j \in J} x_{jr}^{start} = 1 \qquad \forall r \in R \tag{3.10}$$

$$\sum_{j \in J} x_{jr}^{end} = 1 \qquad \forall r \in R \tag{3.11}$$

$$x_{j'r} = x_{j'r}^{start} + \sum_{j \in J} x_{jj'r} \qquad \forall j' \in J, r \in R \tag{3.12}$$

$$x_{jr} = x_{jr}^{end} + \sum_{j' \in J} x_{jj'r} \qquad \forall j \in J, r \in R \tag{3.13}$$

$$x_{jjr} = 0 \qquad \forall j \in J, r \in R \tag{3.14}$$

Mobility and Workspace Constraints: A robot cannot be assigned to a job unless the robot is a valid machine for that job (3.15). It must also have access to the job, either having

it within its local reach or having the ability to move around the assembly environment (3.16):

$$x_{jr} \leq v_{jr} \qquad \forall j \in J, r \in R \tag{3.15}$$

$$x_{jr} \leq g_{jr} + m_r \qquad \forall j \in J, r \in R \tag{3.16}$$

Timing Constraints: Finally, timing constraints are necessary to ensure that the jobs are started and completed in a valid manner adhering to the limitations for the assembly. Jobs can not be completed until after all of the operations in the job are completed (3.17) and the completion time of a job must equal the start time plus the time it takes to process the job (based on the assigned robots) (3.18). The start time of a job can not be less than the completion time of the previous job worked on by that robot plus the amount of time it took the robot to get to the current job (3.19). Additionally, a job can not be started until all the jobs prior to it in the precedence DAG are completed (3.20). A robot can not finish working on a job until its operation is completed (3.21), the time a robot starts moving to a new job can not be before it finishes the job it is working on (3.22), and it can not complete the travel time until the amount of time to move between jobs has passed (3.23). Finally, the overall project completion time (the makespan) can not be less than the largest job completion time (3.24). As noted in Section 3.2.2, a very large constant (L) can be used to turn off constraints when applicable. This is common practice for MIP models since it shifts the value of one side of the constraint based on the configuration of a binary variable. The two constraints using this technique along with the other timing constraints take the

form:

$$c_j \geq c_{jpo} \qquad \forall j \in J, p \in P_j, o \in O_{jp} \tag{3.17}$$

$$c_j \geq s_j + (T_{jpor})x_{jpor} \qquad \forall j \in J, p \in P_j, o \in O_{jp}, r \in R \tag{3.18}$$

$$s_{j'} \geq c_j + S_{jj'r}x_{jj'r} - L(1 - x_{jj'r}) \qquad \forall j \in J, j' \in J\backslash\{j\}, r \in R \tag{3.19}$$

$$s_{j'} \geq c_j \qquad \forall (j', j) \in \mathcal{A} \tag{3.20}$$

$$c_{jr} = c_j \qquad \forall j \in J, r \in R \tag{3.21}$$

$$s_{jj'r} = c_j - L(1 - x_{jj'r}) \qquad \forall j \in J, j' \in J\backslash\{j\}, r \in R \tag{3.22}$$

$$c_{jj'r} = c_j + S_{jj'r}x_{jj'r} - L(1 - x_{jj'r}) \qquad \forall j \in J, j' \in J\backslash\{j\}, r \in R \tag{3.23}$$

$$c_{max} \geq c_j \qquad \forall j \in J \tag{3.24}$$

3.3 Experimental Validation

As a validation for the SAPD and the MIP schedule generation method, a multi-robot assembly problem to build an arch structure was developed. This validation included a three part experiment set to evaluate three different factors. The first experiment takes the optimal schedule that was generated by the MIP and compares it to a hardware implementation following the optimal schedule to evaluate how well the scheduler models the assembly and to provide insight for improvement in the MIP formulation. The second experiment compares the hardware implementation of the optimal schedule against an alternative allocation policy that might be used in an in-space autonomous assembly problem where an autonomous scheduler has not been developed. Finally, the third experiment attempts a reschedule in the middle of the assembly to replicate a realistic condition where something may go wrong and the scheduler will need to be rerun to determine a new set of robot to task allocations

to finish the assembly.

3.3.1 Assembly Problem

The assembly project used in this work consists of taking seven blocks from a storage area, bringing them to a staging area where three subassemblies are made into two columns and the arch crossmember. Next, the three subassemblies are moved to the final assembly location where they are assembled into the finished arch. The workforce of these experiments consisted of two teleoperated robots. It should be noted that, while this system has been developed with full autonomy in mind, the scope of this experiment set is the evaluation of the optimization process. In this scope, teleoperated units are more than adequate to validate the optimization formulation. The following sections will fully describe the different elements in the assembly and the three experiments.

Workspace: The assembly workspace for this experiment took place in a $200in \times 110in$ rectangular space. As pictured in Figure 3.1 Panel A, there are two storage bays where the parts for the columns and horizontal crossmember are stored respectively. Both types of subassemblies (column and crossmember) have their own staging areas where they are assembled before they are brought to the final assembly area to be combined into the completed arch. In addition to these five locations, each of the robots have a starting location defined in the workspace. Table A.1 in Appendix A.1 provides the SAPD formulation for each of the seven reference locations.

Components: The components in this project consist of the seven blocks that make up the arch structure. There are two large blocks (lb1 and lb2) that will be combined with two medium blocks (mb1 and mb2) to construct the two column subassemblies. The final three

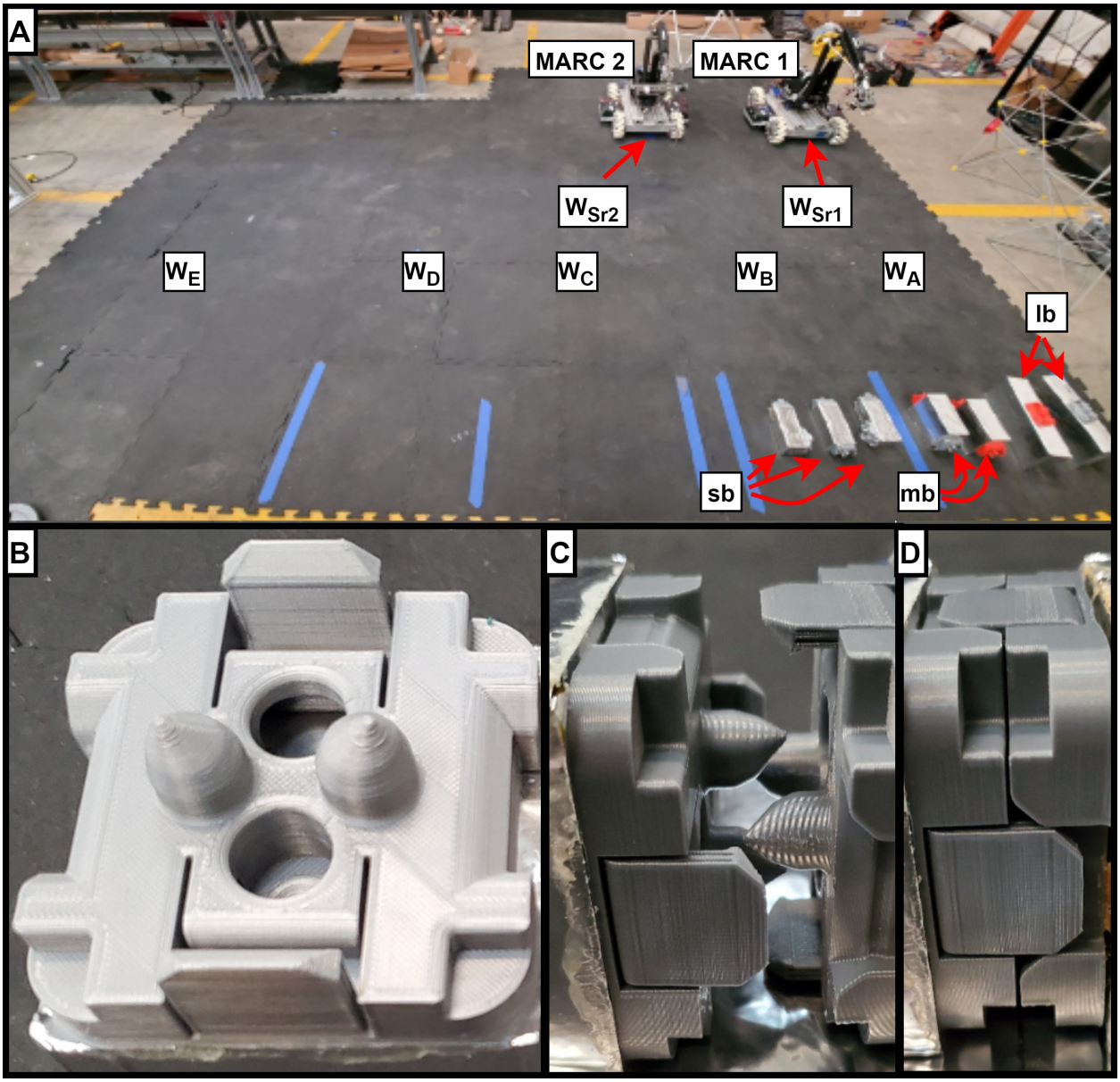


Figure 3.1: Panel A: Arch assembly workspace with location, component, and robot labels, Panel B: One side of the androgynous connector, Panel C: Both sides of the androgynous connector about to be connected, Panel D: Androgynous connector in the connected state.

blocks, two small corner blocks (sbc1 and sbc2) will be combined with one small middle block (sbm1) to make the horizontal crossmember. The SAPD representation for each of these components is given in Table A.2 in Appendix A.2.

Joints: In this assembly project, all of the joints are of a snap connector type. Pictured in Figure 3.1 Panels B-D, these clips are androgynous and include self-aligning posts that help fine tune the alignment after a course alignment takes place. After the course alignment, force along the clip face axis will snap two clips together. There are six of these joints in the entire assembly, each of which is defined in Table A.3 in Appendix A.3.

Robot Workforce: The robot workface consists of two Mobile Assembly Robotic Collaborator (MARC) units designed custom in the Field and Space Experimental Robotics (FASER) laboratory. For the experimental work reported here, the teleoperators for each MARC remained constant to reduce the chance of processing time variation that might be introduced due to different skill levels or a difference in control interface familiarity. To model the processing time for different tasks, each task type was repeated thirty six times by each operator / robot to form a distribution. The expected value of each distribution was used to inform the MIP processing time and setup time information in the model. The distributions for each of these processing times will be discussed in Section 3.3.3. The expected values, along with the rest of the robot formulation description are shown in Table A.4 in Appendix A.4.

Assembly Constraint Formulation: To ensure a valid assembly, components could not be assembled into subassemblies unless they were in their designated staging areas. In a similar fashion, the subassemblies could not be assembled into the final structure until they were in the final assembly area. The only robot to job restrictions were those applying to

the two start jobs. These start jobs were used to insure that the robots started in the right positions. Additionally, there were no continuity constraints in this assembly problem. Four different job types were used in this assembly: idle start (S), move part (M), connect part and connect assembly (C). To complete these jobs, eight different operations were defined: idle, locomote, connect component, co-op connect component, connect subassembly, co-op connect subassembly, pick & place, and finally place. These operations are defined as:

- **idle:** is a very short waiting time for a robot. This is only used to initialize the robot at the correct starting location.
- **locomote:** is the motion contribution a robot can make. This is the only operation that is a velocity and is utilized anytime a robot has to move locations.
- **connect component:** is the action of picking up a component and connecting it to another component.
- **connect subassembly:** is the action of connecting both sides of the crossmember to the columns.
- **co-op connect component:** is the action of connecting one component to another with the help of a second robot.
- **co-op connect subassembly:** is the action of connecting one side of the crossmember to one column (each robot connects one of the two sides).
- **pick & place:** is the action of picking up and setting down a component. This is often used in conjunction with the locomotion velocity to describe the time it takes a robot to complete a move job type.
- **place:** is the place portion of the pick & place operation. This is used during the reschedule experiment when the damaged crossmember needs to be set down for repair.

The first of the four types of jobs, idle start, was a first job for each robot to insure they began in the right position. This job is analogous to the robots coming out of storage and for this experiment, it took negligible time to complete this job type. The move job type, as the name implies, was used when a component needed to be moved from one location to another. The move job type is unique in that the scheduler calculated the processing time for a move job as the time it took to complete the robot operation contribution (pick & place) plus the time it took the robot to traverse the distance between the starting and ending points in the job. The connect job type reflected components or subassemblies being connected, with sub-elements of connect component or connect subassembly. Connect component represented the time it took a robot to place and engage the clips between two components. Connect subassembly, like connect component, involved the robot connecting the clips between blocks. However, in the subassembly type, the time includes the robot having to connect both sides of the crossmember. Tables A.5 and A.6 in Appendix A.5 provide a list and description of the different jobs that make up the assembly project and how each one could be completed, respectively. The DAG describing the precedence constraints, the initial state of the assembly, and the final state of the assembly are all shown in Figure 3.2.

3.3.2 Experiment Implementation

As noted above, the processing time information for each of the operations in the assembly project was generated through thirty six instances of each robot completing each type of operation. Each set of thirty six contained slight variations likely to be seen in the actual assembly project. For example, to evaluate the connect component processing time, combinations of the three different block sizes (small, medium, and large) and three different spacings between pre-connected blocks were used. Thus experimentally generating the pro-

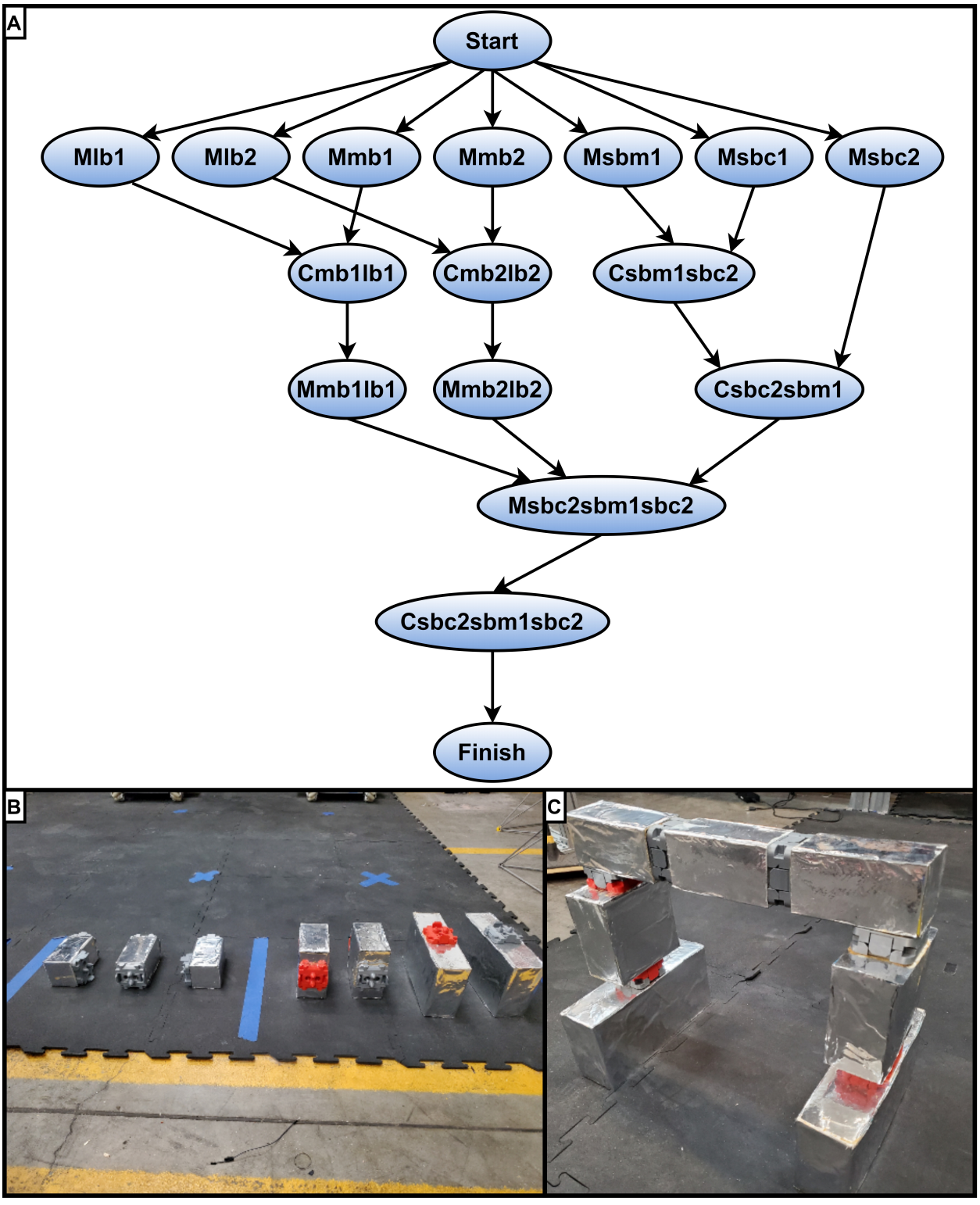


Figure 3.2: Panel A: Precedence DAG for arch assembly project, Panel B: Initial assembly state, Panel C: Final assembly state.

cessing times to inform the SAPD provided the stochastic variation information present due to operator control and slight changes present due to different grips required for the three block types to be modeled. Each robot was controlled by the same operator through all of the experiments to preserve the distributions acquired for the SAPD. The MIP scheduler was implemented using the Gurobi Python API [61] on a Windows 10 computer with 16 GB of RAM and a 2.20 GHz i7 CPU. The solution generated by the MIP was used as the optimal schedule in all three experiments. The optimal schedule was solved for in 9.58 seconds. Figure 3.3 provides a picture of the optimal schedule. The bottom two rows in the graph show which robot worked on what job at what time in the assembly project. The black lines between jobs show when a robot would need to travel between jobs and the blank gaps represent when a robot is waiting to start its next task. The upper portion of the graph shows when each job was processed in the assembly project. The jobs times in the upper and lower portions of the graph are color synced to clarify task allocation when two robots are working at the same time. The projected completion time for the assembly was 679 seconds.

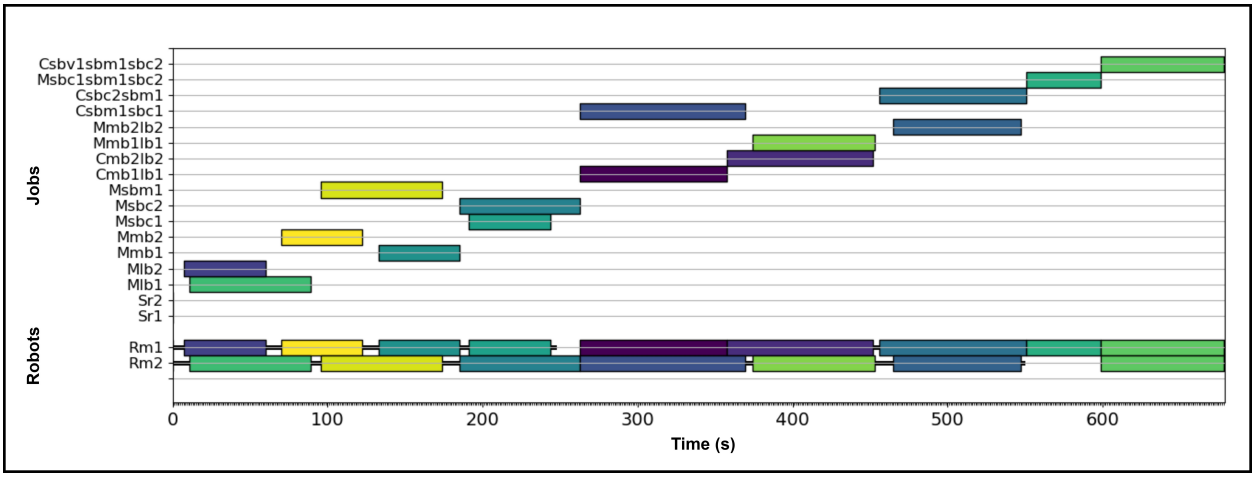


Figure 3.3: Optimal schedule generated by the mixed integer program. The bottom portion of the graph shows the robot to task allocation and the upper portion of the graph shows the task sequencing. The jobs times between the two portions are color coded and spatially synced.

Full Assembly: The full assembly experiment consisted of two complete runs of the assembly project following the task sequence and allocation in the optimal schedule. During the assembly runs, there were a few instances of unexpected errors. In a deployed scenario, these errors would lead to a reschedule and reallocation of the robots. Since the third experiment tests this ability, the other experiments focused on other evaluations. For the first two experiments, if an unexpected error occurred, such as a battery running low or a connection clip breaking, all of the operators were immediately notified and the experiment was paused while the issue was resolved. Once resolved, everything continued from the point where the error occurred, allowing for the down time due to the error to be factored out of the processing times making the comparison between the schedule and the hardware implementation valid. It should be noted that failures in the experiment, such as a gripper needing to be readjusted or a second attempt at making a clip connect, were left in the experiment since they are realistic elements in the assembly problem.

Alternative Policy Task Allocation: To evaluate how the completion time of a hardware implementation of the optimal schedule compares to an assignment method that did not have the task sequencing and allocation analysis capability, an alternative assignment policy was chosen that reflects a realistic strategy to complete a new assembly project. This alternative policy allocated tasks based on their type to a specific robot. MARC 1 was faster at completing the connect component operations than MARC 2 so all of the connection type jobs were given to MARC 1. Similarly, MARC 2 was given all of the move job types. To give this policy the best chance against the optimal policy, the jobs were completed in a way that would minimized the amount of time one robot was waiting on the other. While this changed the task sequencing slightly in the alternative policy hardware implementation, it allowed for the best comparison between the optimal task allocation and the alternative policy allocation.

Replanning Error Correction: To test the ability of this scheduler to reschedule, one of the motivations in the development of the task assignment framework, a failure was forced near the end of the assembly. Once the crossmember arrived at the assembly location, the end of the crossmember was removed from the other two components. This required a reassignment with a new starting configuration reflecting the current state of the assembly. Jobs were inserted to set the crossmember down, repair it, and then finish the assembly.

3.3.3 Results

Full Assembly: Two full assembly runs following the optimal schedule were completed. In the first, part way through the assembly, one of the connection clips broke requiring the experiment to be paused and the component replaced. In the second assembly, an electronics issue occurred also requiring the experiment to be paused. The presence of these errors further highlight the necessity of a system capable of articulating the states needed for repair and reallocating robots autonomously. As described above, both of these pauses were removed from the data in post-processing and the resulting data was verified against the footage.

First Full Assembly Run: In Figure 3.4 Panel A the comparison between the first full assembly run and the optimal MIP schedule is shown. This assembly took 801 seconds to complete, an 18.14% error in overall completion time of the project. A job by job break down of the differences and percent errors between the optimal projected schedule and this hardware trial is shown in Figure 3.4 Panel B. The largest two errors occurred when dealing with the horizontal crossmember. As shown, all of the jobs moving components from the storage to the staging area took a few second less than the predicted time, on average. Alternatively, all of the jobs dealing with the subassemblies took longer than predicted aside from the final job where the two robots connected the horizontal crossmember to the two columns.

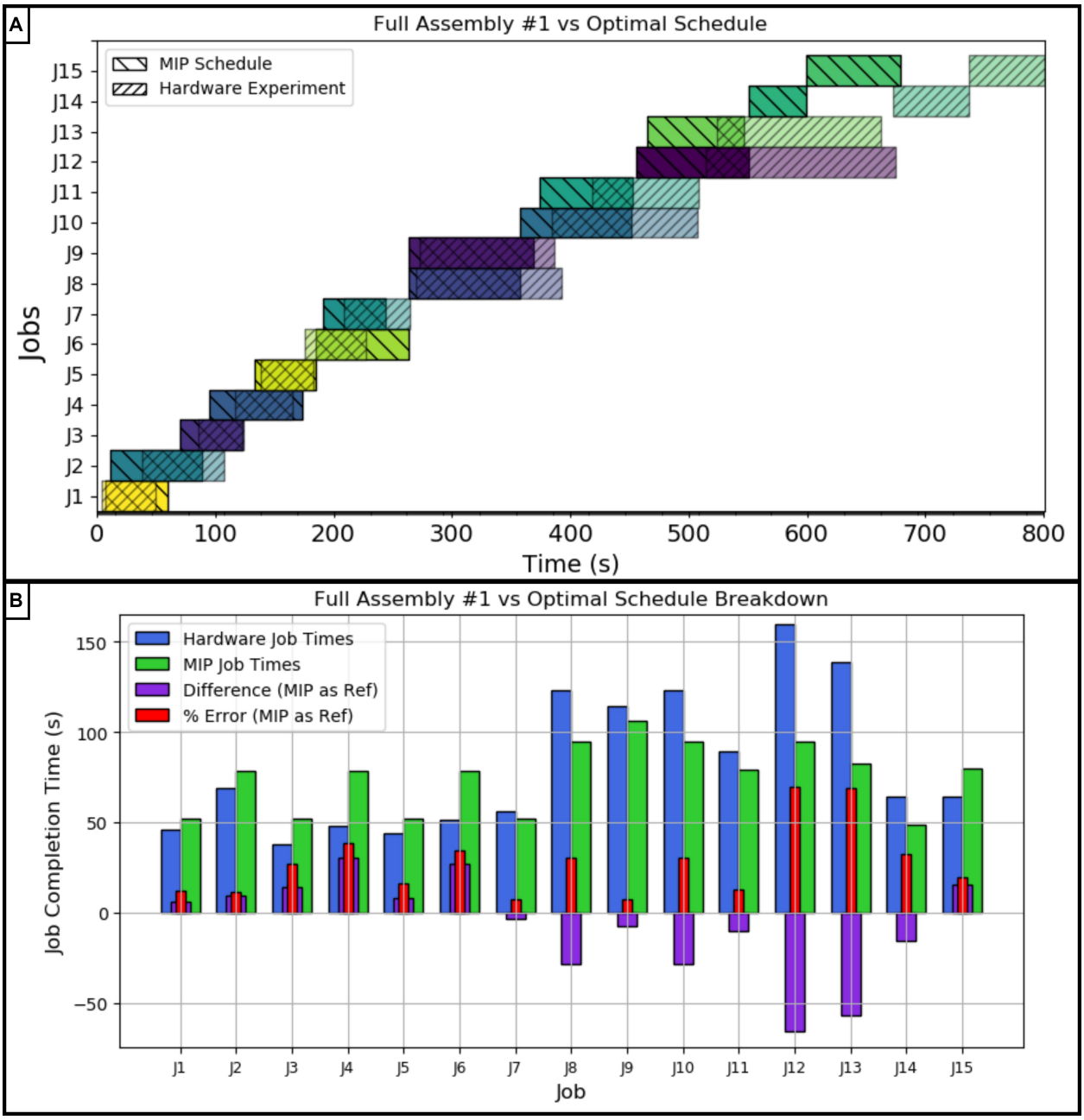


Figure 3.4: Panel A: Hardware vs MIP optimal schedule for the first full assembly run, Panel B: Job by job breakdown with error for the first full assembly run. The job key is as follows: $J1 : Mlb2$, $J2 : Mlb1$, $J3 : Mmb2$, $J4 : Msbm1$, $J5 : Mmb1$, $J6 : Msbc2$, $J7 : Msbc1$, $J8 : Cmb1lb1$, $J9 : Csbm1sbc1$, $J10 : Cmb2lb2$, $J11 : Mmb1lb1$, $J12 : Csbc2sbm1$, $J13 : Mmb2lb2$, $J14 : Msbc1sbm1sbc2$, $J15 : Csbc1sbm1sbc2$.

Second Full Assembly Run: Similar to the first full assembly run, the second assembly took 814 seconds to complete, a 20.06% error from the optimal schedule. Like the first assembly, the schedule comparison shown in Figure 3.5 shows similar error trends where the moving of parts between the storage and staging areas were slightly faster than predicted and the jobs dealing with the subassemblies took longer than projected. In the second assembly run, the largest difference between actual and projected comes from moving the crossmember into place.

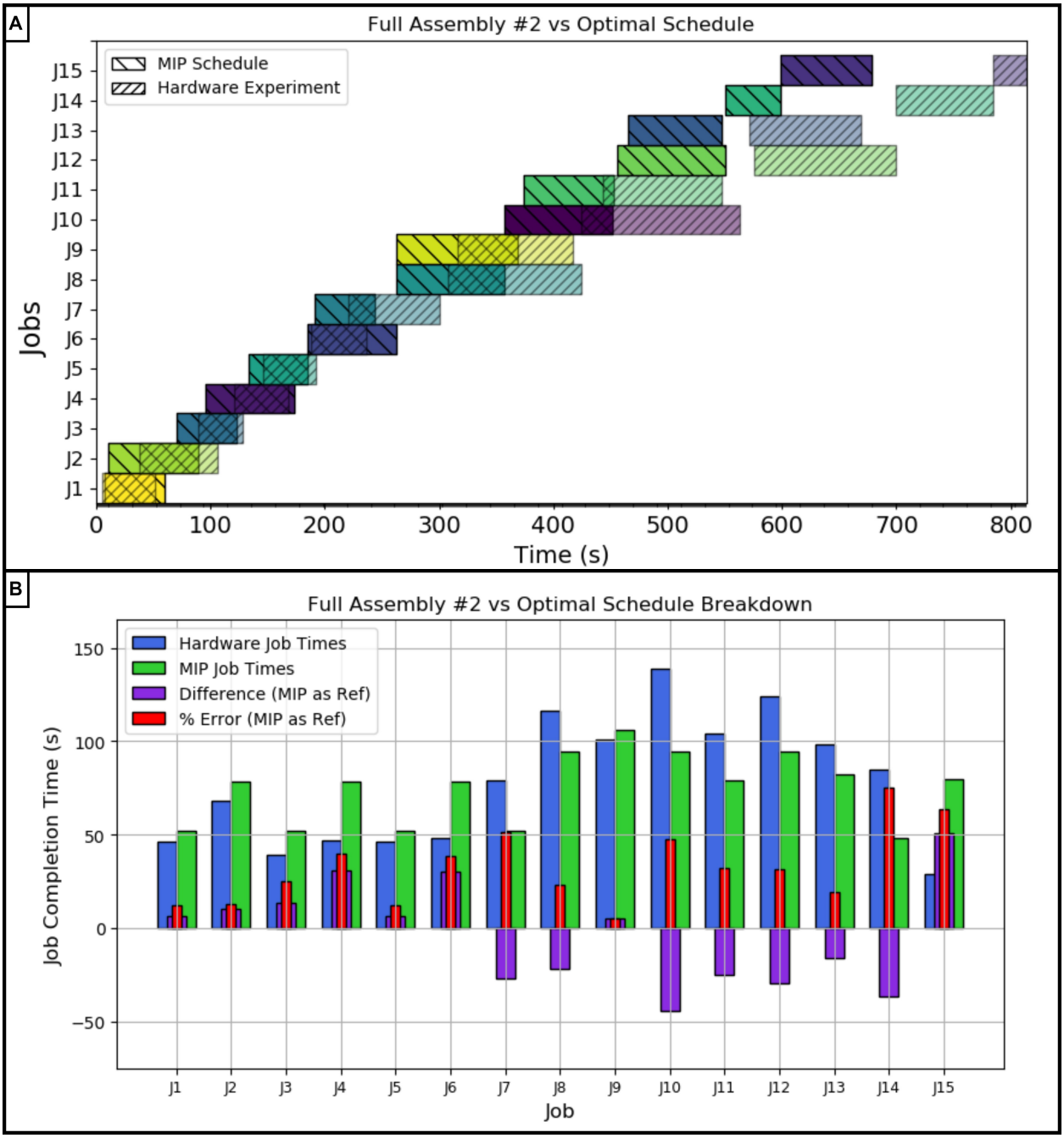


Figure 3.5: Panel A: Hardware vs MIP optimal schedule for the second full assembly run, Panel B: Job by job breakdown with error for the second full assembly run. The job key is as follows: $J1 : Mlb2$, $J2 : Mlb1$, $J3 : Mmb2$, $J4 : Msbm1$, $J5 : Mmb1$, $J6 : Msbc2$, $J7 : Msbc1$, $J8 : Cmb1lb1$, $J9 : Csbm1sbc1$, $J10 : Cmb2lb2$, $J11 : Mmb1lb1$, $J12 : Csbc2sbm1$, $J13 : Mmb2lb2$, $J14 : Msbc1sbm1sbc2$, $J15 : Csbc1sbm1sbc2$.

Alternative Policy: As stated in Section 3.3, the alternative assignment policy where MARC 1 completed only the connecting type jobs and MARC 2 completed all of the move type jobs, was used to evaluate the benefit of using the optimal schedule against an alternative, realistic assignment policy in an autonomous scenario. The job completion time results for this hardware trial are compared against the optimal schedule and the first hardware implementation of the optimal schedule in Figure 3.6 Panel A and Panel B respectively. Even with the reordering of task sequencing to complete the alternative policy in the optimal way, the alternative policy makespan was 901 seconds, yielding an error of 32.89 % from the optimal schedule projection and a 12.48 % error longer from the first hardware run implementing the optimal schedule.

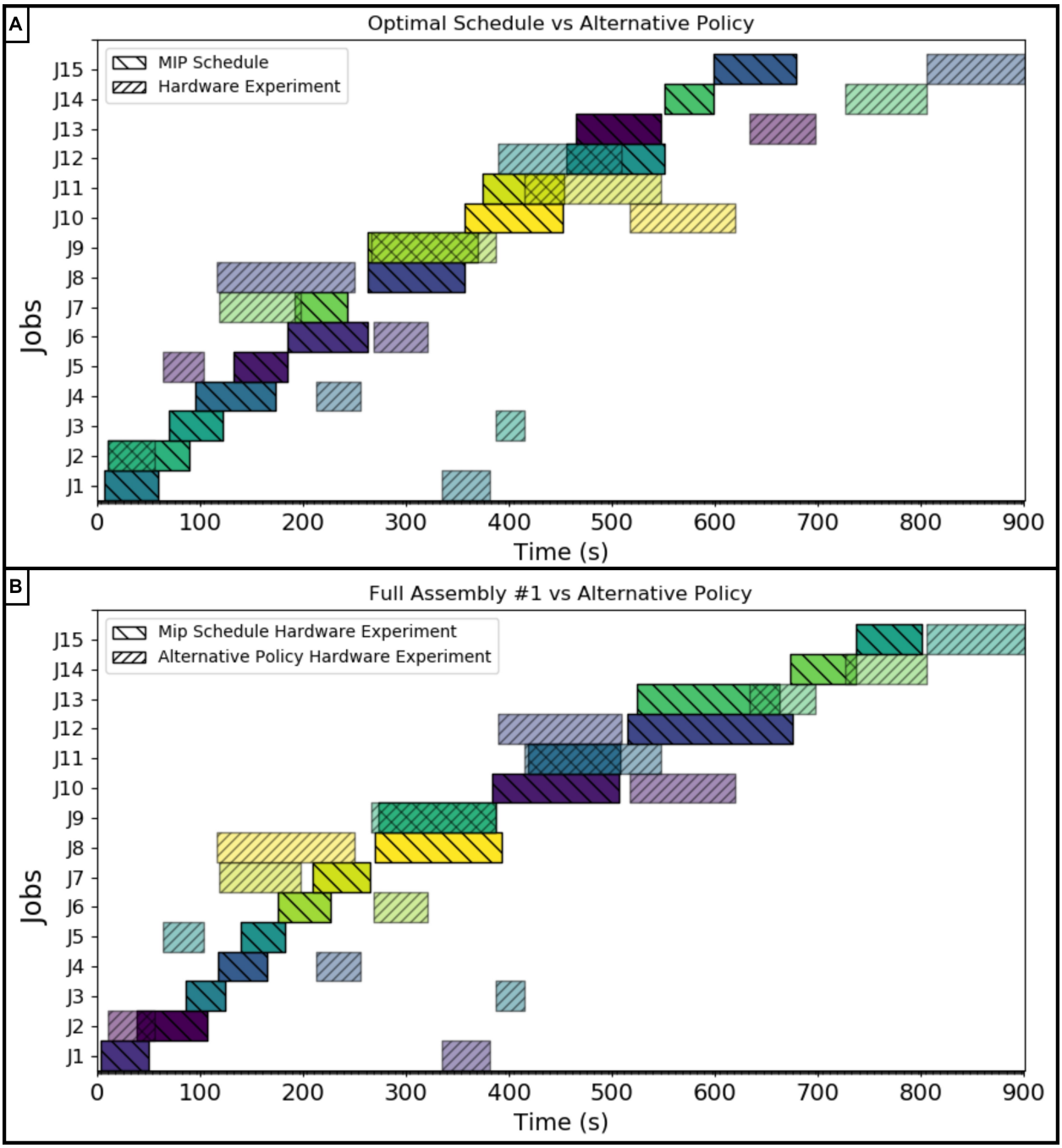


Figure 3.6: Panel A: Hardware experiment following alternative policy compared against the optimal MIP scheduled, Panel B: Hardware experiment following alternative policy compared against hardware experiment following the optimal schedule. The job key is as follows: $J1 : Mlb2$, $J2 : Mlb1$, $J3 : Mmb2$, $J4 : Msbm1$, $J5 : Mmb1$, $J6 : Msbc2$, $J7 : Msbc1$, $J8 : Cmb1lb1$, $J9 : Csbm1sbc1$, $J10 : Cmb2lb2$, $J11 : Mmb1lb1$, $J12 : Csbc2sbm1$, $J13 : Mmb2lb2$, $J14 : Msbc1sbm1sbc2$, $J15 : Csbc1sbm1sbc2$.

Replanning: Using the same computer that generated the original schedule, the reschedule was solved in 0.066 seconds (negligible) time and the assembly proceeded with MARC 1 being retasked (since it was the closest to the damaged part) to set the partial crossmember down, repair the broken joint, and finish the assembly. After the repair portion of the jobs, the new schedule matched the original task allocation, completing final job with both MARCs to attach the crossmember to the two columns.

Processing Times: When applicable, the processing times measured during the hardware experiments were plotted over the distributions of the data used to generate the expected values in the scheduler's processing time entries. Figure 3.7 contains the processing time distributions for the connect component, connect subassemblies, co-op connect component, and co-op connect subassemblies operations for both of the robots. The hardware processing times for MARC 1's connect component appear to be in the upper range of those seen in the data taken for ability property. As shown in Panel A, it took about thirty five seconds longer, on average, than the value used in the scheduler. This longer time corresponds to the higher error in the connect component job types shown in the schedule. In contrast, MARC 2, which only had a few instances of connecting a component throughout the hardware trials, had a processing time very close to the expected value of the input data when completing $Csbm1sbc1$. Panels C and D show only one and no hardware data points for MARC 1 and 2, respectively, since the optimal scheduler used both robots to connect the subassemblies together. The one entry for MARC 1 reflects the alternative policy requiring MARC 1 to handle all of the connection job types on its own. Similarly, the only co-op operation utilized was connecting the subassembly shown in Panel F. The final three operations, locomote, pick & place, and place, show similar results in Figure 3.8. The locomote instances for both robots, shown in Panels A and B, are similar to those used by the scheduler. The hardware processing times have not been corrected for the extra distance the robots had to travel

to avoid colliding (since path planning was not part of this experiment set). The similar distributions between the hardware trials and abilities data indicates that this was not a large contributor to the error seen between the optimal schedule and the results. The pick & place operations, Panels C and D, show that the distribution ranges are about the same but with instances 20 to 30 seconds faster and slower than the expected value of the input data.

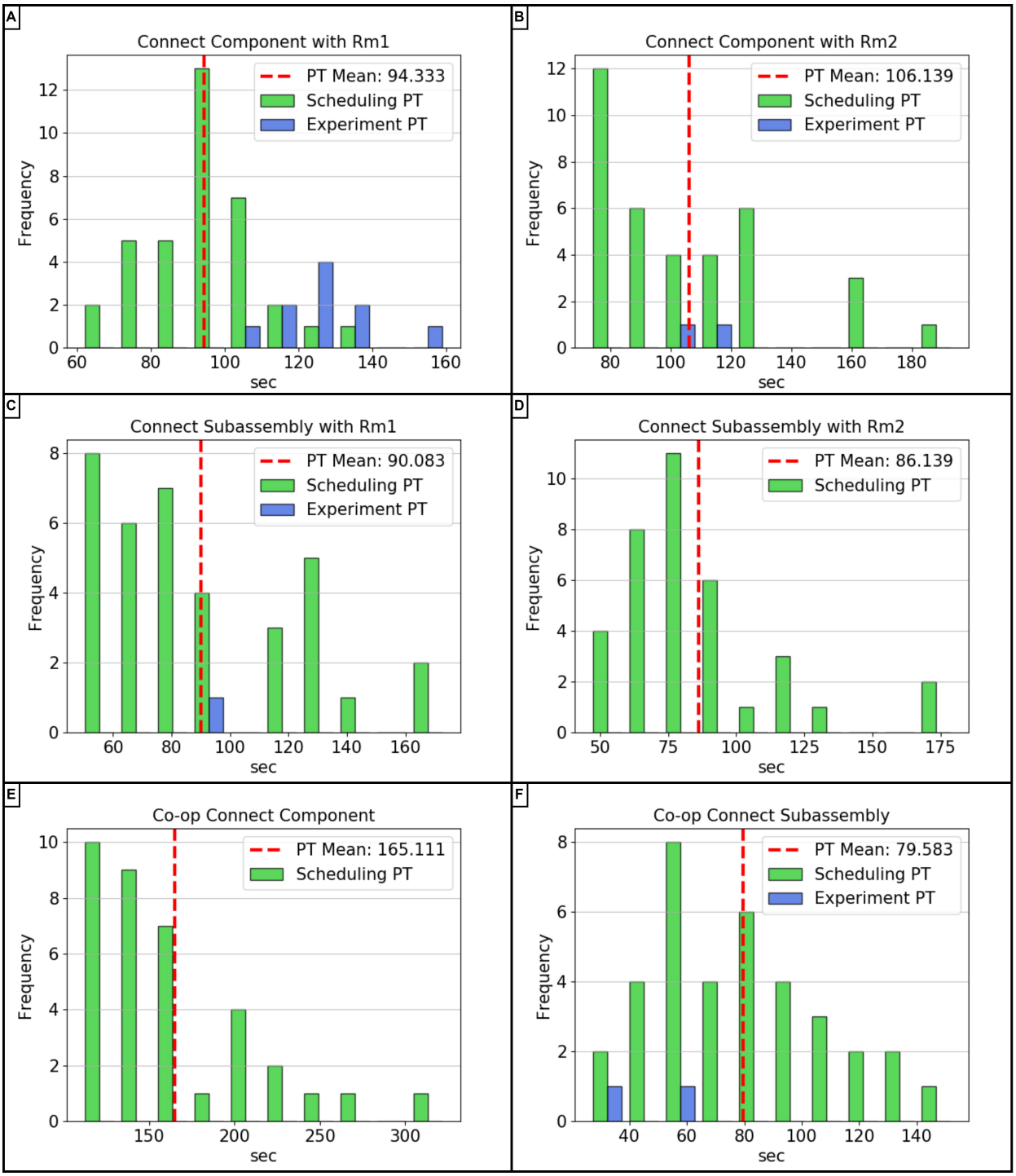


Figure 3.7: Panel A: MARC 1 connect component processing times for scheduler and hardware instances, Panel B: MARC 2 connect component processing times for scheduler and hardware instances, Panel C: MARC 1 connect subassembly processing times for scheduler and hardware instances, Panel D: MARC 2 connect subassembly processing times for scheduler (there were no hardware instances), Panel E: Co-op connect component processing times for scheduler (there were no hardware instances), Panel F: Co-op connect subassembly processing times for scheduler and hardware instances.

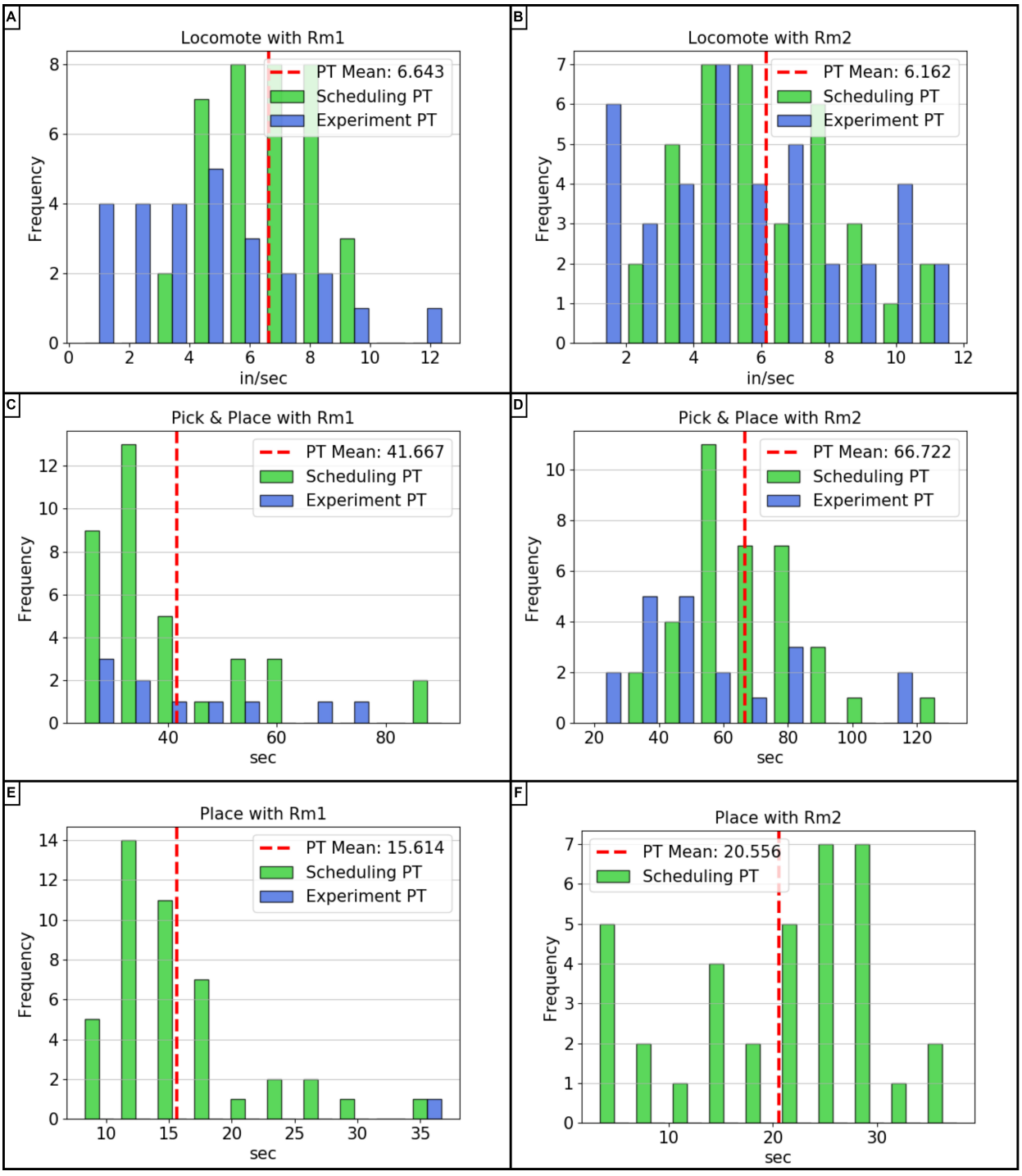


Figure 3.8: Panel A: MARC 1 locomote velocity for scheduler and hardware instances, Panel B: MARC 2 locomote velocity for scheduler and hardware instances, Panel C: MARC 1 pick & place processing times for scheduler and hardware instances, Panel D: MARC 2 pick & place processing times for scheduler and hardware instances, Panel E: MARC 1 place processing times for scheduler and hardware instances, Panel F: MARC 2 place processing times for scheduler (there were no hardware instances).

3.3.4 Discussion

The developed SAPD was able to successfully articulate all of the elements present in the assembly problem that were important to task assignment and successfully encode the constraints in a way that could be passed to a schedule generator to generate task sequences and allocations. The MIP schedule generator was then able to take these elements and constraints to solve for an optimal schedule, which proved to be more optimal to follow than an alternative, less informed, viable scheduling policy. The below sections will discuss advantages and limitations present in the formulations and will end with future work suggestions.

Advantages: The SAPD is capable of describing the elements present in an assembly problem. It has a flexible formulation that allows it to adapt between simple, small scale, assembly projects, up to large, stochastic projects. The ability to record both state and property information allows it contain information that may be needed by other elements of an autonomy suite used in in-space assembly scenarios. The MIP formulation proved to be capable of modeling distance considerations, continuity constraints, and the other core constraint requirements to ensure a valid assembly schedule. By the nature of the branch and bound solving method, used for MIP formulations, it was able to guarantee an optimal schedule based on the input data and contained the ability to give a measure of how far from the optimal solution the generated schedule might be. At the experimental scale presented in this work, schedule generation was very fast, making it usable for rescheduling, an important consideration for in-space applications where the delay or lack of communication can make teleoperated rescheduling impractical or impossible.

Limitations: Using the MDP formulation in the SAPD may reach intractable state numbers for very large assemblies. While this is not a direct limitation to the formulation,

additional work may need to be done to determine the best way to approximate the states or downsample to the a smaller, important state set. This is additional work that would need to feed the problem definition, adding another capability requirement in the overall autonomy system. The MIP formulation, in its current form, does not take into account the spatial footprint of the robots. While it did not greatly impact the overall assembly runs above, it did cause some of the robot interactions to go unaccounted for in the model. One possible mitigation is to factor in path routing in the MIP. This, however, leads to the second limitation. The MIP treats the whole assembly problem all at once, instead of solving it temporally from beginning to end. If a change is made, the overall assembly changes and the solving process needs to start over. While this can be mitigated by feeding in the previous solution to "warm start" the solver, it could cause limitations if the solving time is too large. This is a potential concern for large projects since the addition of jobs or robots will increase the problem space significantly thereby increasing the computation time required to produce a schedule. Future experiments need to be done to determine the scale limitations of the MIP formulation.

3.4 Summary of Contribution

The mixed integer programming formulation presented in this chapter addresses the second research question inquiring about a scheduling formulation that can provide the optimal schedule or one that is some quantifiable measure from optimal.

The mathematical formulation presented here takes the information from the SAPD and describes it in a form that state of the art solvers can use. By using the branch and bound solver method, an optimal schedule can be proven or the gap between a linearly relaxed solution, one that generates a completion time faster then a valid solution, and a known

valid solution. This method provides a bounding within which the optimal solution must fall and can be used to quantify a worst case measure on how far the generated schedule is from the optimal schedule.

The experiment included in this chapter demonstrate this scheduling methods use in an assembly problem to generate the initial schedule and to provide a rescheduling to correct an error that occurred.

Chapter 4

Task Assignment: Stochastic Formulation

To address some of the limitations present in the mixed integer programming approach in Chapter 3, a heuristic search method is needed to allow for continual sampling from processing time distributions. A formulation with this capability will better utilize the stochastic information if it is known in an assembly scenario. This type of solution method formulation also has the potential to handle larger scale projects by providing solutions that have been optimized to the scheduling and allocation problem relatively quickly. However, this may come at a cost of the solution not being optimal (or at least not provably so without also using the MIP formulation developed in Chapter 3).

4.1 Research Gap

Different methods have been developed and applied to the FJSP problem. These include linguistic based approaches [17], tabu search methods [18, 120, 122], simulated annealing [41, 43], and evolutionary algorithms such as genetic algorithms [34, 110] to name a few. Of the large variety applied to this problem set, genetic algorithms work well due their robustness to handle discrete and noisy objective functions [62].

Many of the genetic algorithms used in scheduling complex planning problems such as robotic

path planning [105], road intersection planning [42], trainer scheduling [64], and network sensor scheduling [62] require the use of variable length chromosomes. However, this variability in dimensionality adds difficulty and requires specialized genetic operators to propagate the optimization [64, 84].

For the FJSP-IPP problem specifically, there have been several proposed GA solutions [78, 123, 149]. However, many of these can encode infeasible or illegal solutions which hinders the efficiency of the search [67]. As such, additional operators often have to be added to try and repair infeasible solutions raising the complexity of the process and increasing the computation time for each generation [10].

The presence of continuity constraints in the autonomous robotic assembly problem is not present in the current literature and only furthers the complexities of possible infeasible solutions due to the vast number of ways tasks can be allocated in a way that violates this particular constraint type. To address this, a novel formulation expanding on existing state-of-the-art techniques such as dynamic decoding [67] and constraint preserving genetic operations [92] is proposed. This formulation provides an encoding that minimizes the possible rise of infeasible solutions to a single decision component and provides a framework capable of modeling the continuity constraint in addition to already existing constraints such as precedent constraints and process plan considerations.

4.2 Genetic Algorithm

A genetic algorithm is an optimization method first developed by John Holland and his collaborators [144] based on the concepts of genetics and natural selection. The algorithm functions by first generating an initial population of chromosomes that represent a sampling of different instances of the problem being optimized. This population is propagated

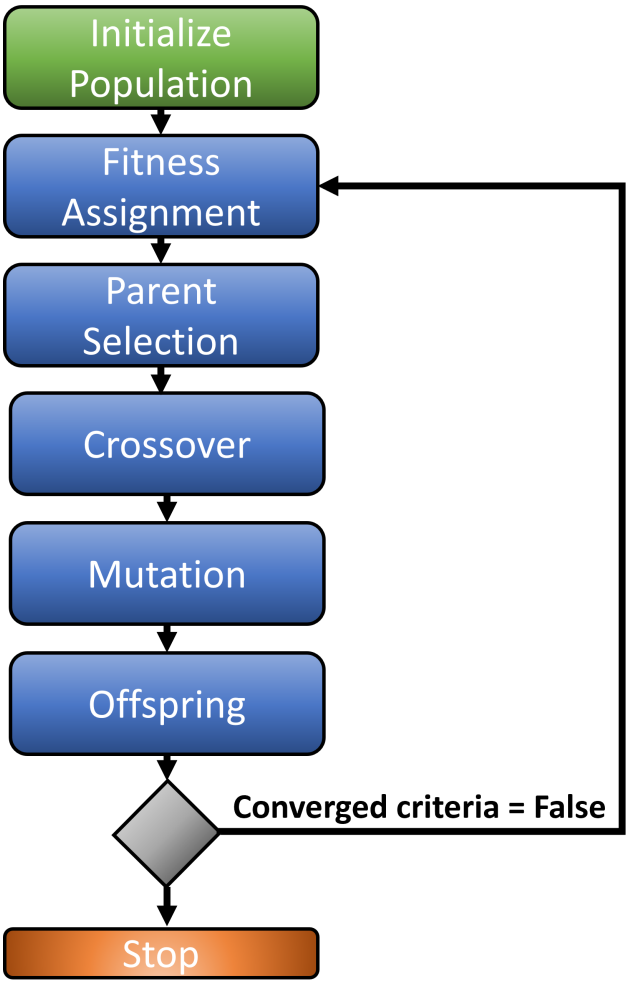


Figure 4.1: The general architecture of a genetic algorithm.

through multiple generations using processes such as crossover and mutation to seek the best individual. Throughout this process, a fitness criteria is used to evaluate how well a chromosome will survive. Based on this information, a selection process is made and the cyclical generation propagation continues [136]. This basic anatomy is pictured in Figure 4.1. The explanation and implementation of each step are discussed in the following sections.

4.3 Formulation

The formulation described here presents a novel encoding and decoding method to account for continuity constraints present in a JFSP-IPP problem. This architecture encodes the problem using methods from recently published papers extended to encapsulate this new constraint type while preserving the previous constraint types. To aide in the understanding of this formulation (decoding in particular) an example problem will be utilized.

4.3.1 Example Problem

The example project focuses on assembling a stack of blocks. They must be moved from a storage area to an assembly area, stacked, and then fastened into place. Pictured in Figure 4.2, this assembly project utilizes three robots (two of the gripper type and one with a welder type) to complete the project.

In this project, four different job types are used. The start job type (S) is used to represent the starting job for each robot. This can be thought of as the robots coming out of storage and preparing to operate. As such, these jobs most be completed before any other job can occur. The next three job types apply to the blocks in the assembly and correspond to the tasks of moving (M), aligning (A), and fastening (F) the blocks together. Naturally, there is a precedence constraint set that must be followed for these jobs to succeed. Once an alignment takes place, the same block that was aligned must be held in place while it is being attached. This leads to the new type of constraint, the continuity constraint. This constraint spans across the aligning and fastening jobs, requiring the same robot to complete the grasping operation in both cases without working on any tasks in-between. Figure 4.3 contains a representation of these jobs, their process plans, operations, and the constraints using the formulation defined in Chapter 2.

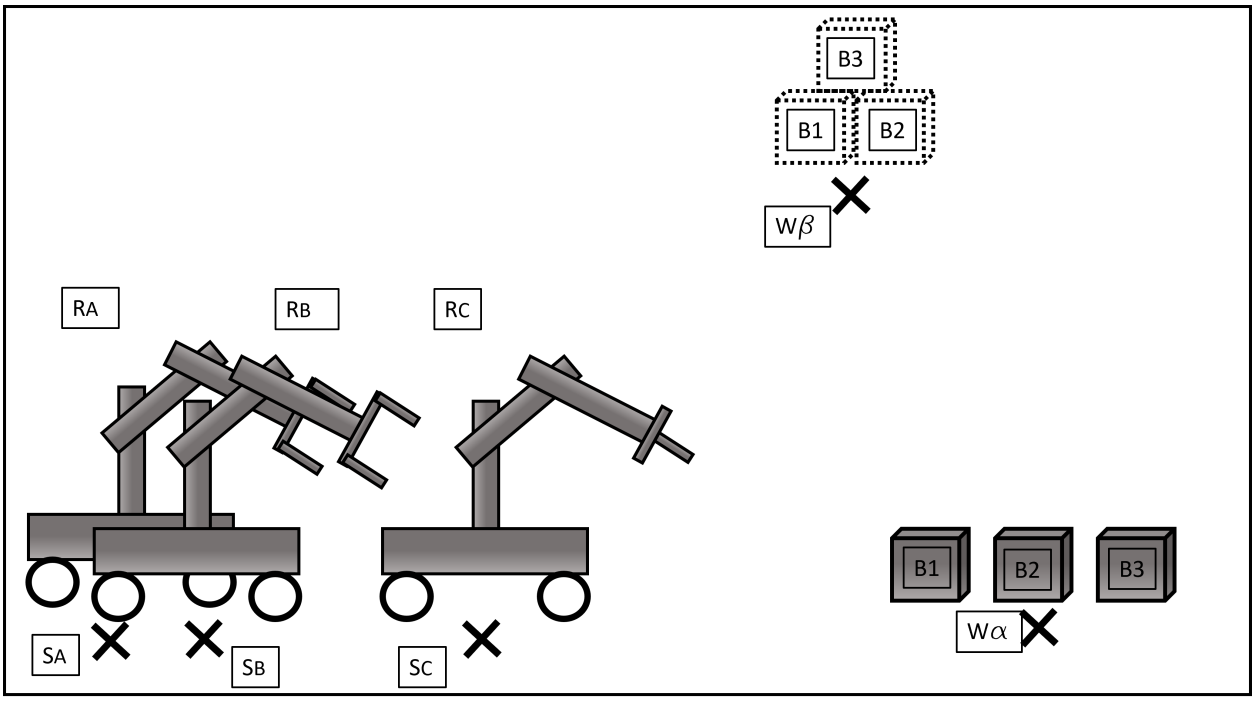


Figure 4.2: Workspace consisting of three robots and three blocks.

Jobs	Process Plans	Operations	Valid Machines
J_{SA}	P_0	O_1: grasp	R_1
J_{SB}	P_0	O_1: grasp	R_2
J_{SC}	P_0	O_1: grasp	R_3
J_{M1}	P_0	O_1: grasp	R_1 R_2 R_3
J_{M2}	P_0	O_1: grasp	R_1 R_2 R_3
J_{M3}	P_0	O_1: grasp	R_1 R_2 R_3
J_{A12}	P_0 P_1	O_1: grasp O_1: grasp O_2: grasp	R_1 R_2 R_3
J_{A23}	P_0 P_1	O_1: grasp O_1: grasp O_2: grasp	R_1 R_2 R_3
J_{F12}	P_0	O_1: grasp O_2: weld	R_1 R_2 R_3
J_{F23}	P_0	O_1: grasp O_2: weld	R_1 R_2 R_3

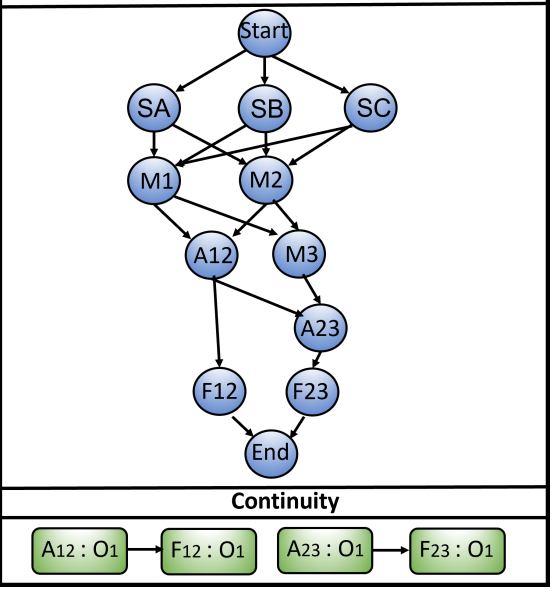


Figure 4.3: Each job in the project has a set of process plans, operations, and valid machines. The project also contains precedence and continuity constraints.

4.3.2 Encoding Strategy

In this type of problem there are several different types of decisions being made. The first is a selection for how each job will be completed, the selection of a process plan for each job. This selection affects the third decision type, the machine assignment, in that the number of machines required to complete a job will change depending on how long each machine's contribution will take. An example of this would be if the operation required when two robots work together only takes half as long as an operation that would require a robot to complete the job alone. This type of collaboration requirement consideration is not present in the literature. Rather, each machine's task is considered independent and the robots can come and go between their contributions. The complexities of the allocation decision will be addressed in a future section. The second decision type, the sequencing decisions, will affect the machine assignment decisions. Depending on the job sequencing, certain robots may no longer be free to work on jobs that they typically would be valid candidates as a result of the continuity constraints. The interconnected nature of these elements are addressed using dynamic decoding, which will be described in the following sections.

Three Part Chromosome: The three different decision types present in this problem are encoded using three different chromosome parts. Pictured in Figure 4.4, the first chromosome portion encodes the process plan selection. In this formulation, there is a gene for each job and the value of that gene will represent which process plan is being used. The second chromosome portion encodes the job sequencing. Each gene position represents the priority given to the job in that location, identified by the value of the gene. The job in the left most gene will be given priority over those to the right, moving sequentially down the chromosome portion. The final chromosome portion encodes the machine allocation where each gene location represents an operation in a job and the value within the gene represents the machine

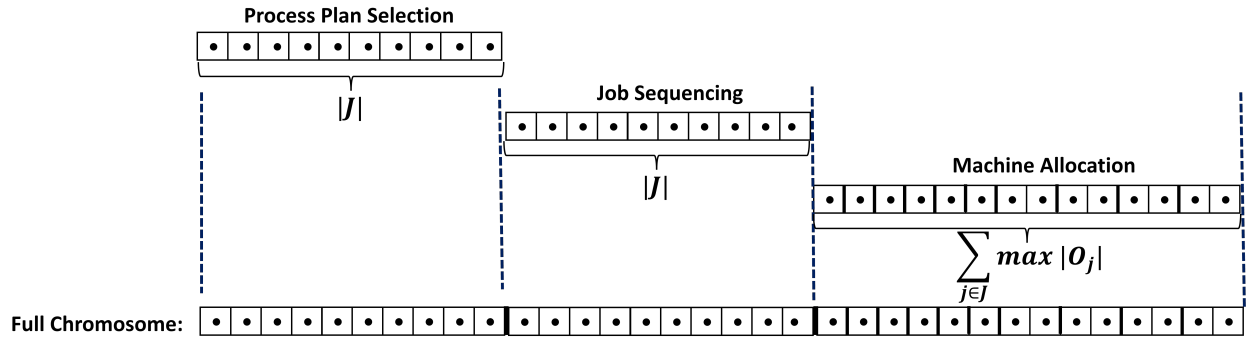


Figure 4.4: A full chromosome consists of three parts: the process plan selection, the job sequencing, and the machine allocation.

the operation is being allocated to.

The presence of varying numbers of process plans, each potentially containing different numbers of operations plus the continuity constraints ruling out certain machines based on previous assignments complicates the encoding process. A standard encoding approach would represent each operation as a gene in the chromosome for task allocation. However, since the operations can change, this would lead to complications when selecting and breeding chromosomes, requiring complex methods to deal with the variable size to still provide the desired exploration of the solution space. As stated above, this is dealt with using dynamic decoding. To start, a system of value mapping must be used in the process plan and machine allocation chromosome portions. This will be described in the next section.

The encoding of the job sequencing is done by mapping each job to an ID number. Since each job can only be processed once, the job sequencing chromosome portion can only contain one entry of each ID for all of the jobs. This chromosome portion, unlike the processing plan and machine allocation portions, can more easily be inspected for infeasible solutions. This occurs from the ID values (jobs) being placed in a sequence that violates the precedence constraints. A method dealing with this chromosome portion in the population initialization, crossover, and mutation steps are designed to take this into account. The details of which

will be explained in depth in Sections 4.3.4 - 4.3.7 respectively.

Max Value Mapping: To address the issue where different genes may need to represent a different number of options within the same chromosome part, a max value mapping is used. Similar to [67], a max possible value is found for all of the genes in that chromosome portion. The genes' values can then take any number between 1 and that max value. When decoding, that number range can then be remapped to the valid range for that gene. This will be discussed in more depth below. The two chromosome portions that require this method are the process plan selection, since each job (and therefore gene) may have a different number of process plans, and the machine allocation, since the possible machines that can be assigned to an operation may change. The max value that a processing plan gene can take is defined as:

$$maxP = \mathbf{max}(|\mathbf{P_j}|) \quad \forall \mathbf{j} \in \mathbf{J} \tag{4.1}$$

Where $|P_j|$ is the number of process plans for job j. Therefore, the processing plan section of the chromosome is a made up of $|J|$ genes where each gene with a value from the range $[1, maxP]$. The machine allocation portion of the chromosome is defined in a similar fashion. The value each gene can take has an upper bound of the most machines that can be assigned to a job, that is the most valid machines that can be present (V_j) for job j. This results in a gene range of $[1, maxM]$ where $maxM$ is defined as:

$$maxM = \mathbf{max}(|\mathbf{V_j}|) \quad \forall \mathbf{j} \in \mathbf{J} \tag{4.2}$$

This feature of the encoding is important since it allows all of the genes in a particular chromosome subsection to use the same number ranges for the genes present in that section. As will be discussed in a following sections, the crossover and mutation methods are done

by chromosome subsection. This mapping allows for easy interaction between all of the chromosomes portions of the same type.

4.3.3 Decoding Strategy

Due to the interconnected nature of the three chromosome parts to create a valid solution, a dynamic decoding process is used which utilizes the value remapping capability described above. Figure 4.5 contains an example chromosome of a solution to the example problem given above. The processing plan portion of the chromosome is decoded by solving for what value contained in the gene represents for that particular job. This decoding is done using Equation 4.3 where p'_i is the value present in gene i representing the process plan for the job represented in that gene. Using the max possible value in the chromosome portion, $maxP$, and the highest number of process plans that exist for gene i's job, $|P_i|$, the process plan p_i is decoded and assigned for that job.

$$p_i = \left\lceil \frac{p'_i}{maxP} \times |P_i| \right\rceil \; i = 1, \ldots, |J| \tag{4.3}$$

Decoding the sequencing chromosome is done by sweeping from left to right through the chromosome, referencing the job represented by the ID value present in the gene. It is important to note that the sequencing is the preference in assignment. This can be thought of as a required ordering for the jobs in a given machines assignment. It does not explicitly require that a job represented by a previous gene is completed before a gene later in the chromosome. This is implied in that the job sequencing will follow precedence constraints defined in the problem and it will be enforced when calculating the fitness (makespan) of the solution.

Decoding the machine allocation portion of the chromosome is the most complex portion of

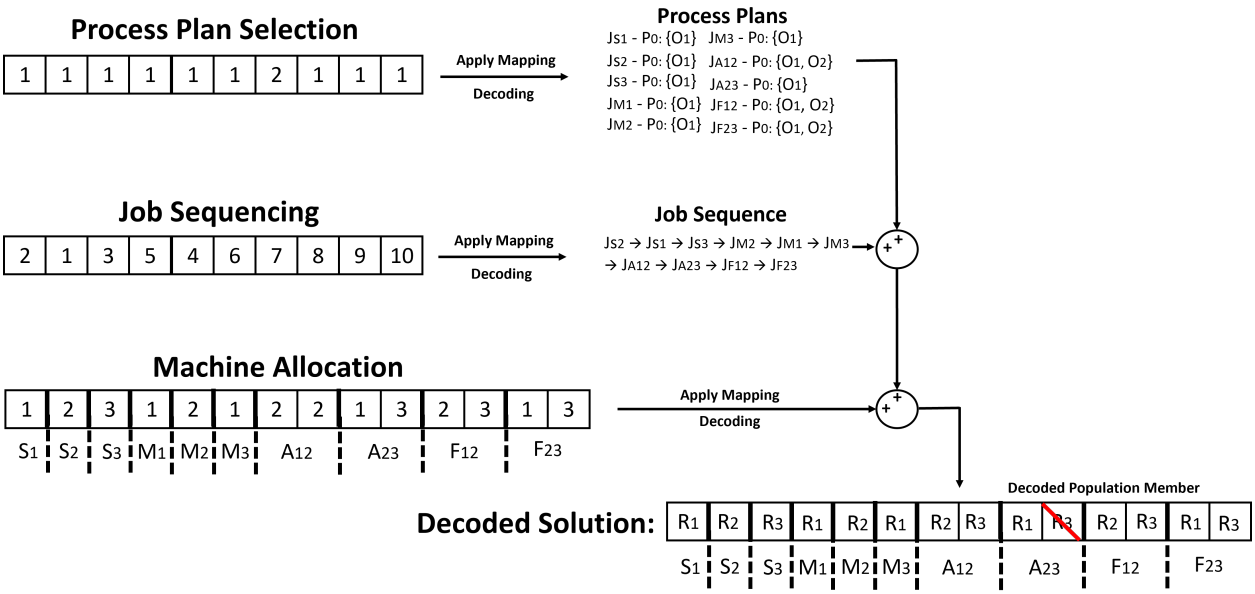


Figure 4.5: The decoding process first deciphers the process plan and job sequencing portions of the chromosome. The decoded result is used to inform the dynamic portion of the machine allocation decoding resulting in a fully decoded solution.

the decoding process. As indicted in Figure 4.5, both the processing plan and job sequencing inform the decoding of the machine allocation. First, based on the process plan selected for a job, the genes representing operations that will not be used are removed (represented as a red strikethrough in Figure 4.5). Using the order defined in the job sequence chromosome portion, each valid operation is mapped to a machine using a method similar to the process plan mapping:

$$m_i = \left\lceil \frac{m'_i}{maxM} \times |V'_i| \right\rceil \; i = 1, \ldots, |J| \tag{4.4}$$

where m'_i is the gene value, V'_i is the set of valid machines for the job represented in gene i, and m_i is the correct machine being assigned. The functionality of V'_i is very important in the context of dynamic decoding. As part of the continuity constraints, when a robot is assigned to the first operation in a continuity it can not be used on any other operations until it has been assigned to the second operation in the constraint. To account for this, V'_i dynamically

varies between the original set of machines that were viable, V_i, and the amount of machines that are valid taking into account how the previous machines are allocated based on the earlier genes in the chromosome. It is this feature that adds the functionality of continuity constraints into the formulation providing a capability not yet seen in the literature. Using this process, the chromosome can be fully decoded into the solution and evaluated for fitness.

4.3.4 Initial Population

Since a genetic algorithm optimizes by crossing and mutating chromosomes, the optimization process must begin with an initial population. For complex systems where there is a possibility of infeasible solutions, this initialization process can be difficult. As discussed above, the encoding process developed for this work was designed to minimize the ability to produce infeasible solutions. However, some of the methods used in this optimization process prevent infeasible solutions by preserving viable solutions (this is the case for the sequencing chromosome portion). After the three portions of a chromosome are initialized, they are combined and decoded to make sure they are valid. If they are not, the chromosome is rejected and the initialization process starts over for that chromosome. The below sections will discuss the initialization process for the three different portions of the chromosome.

Process Plan: Initializing the process plan chromosome portion is the simplest initialization of the three parts. To initialize, a chromosome of the length $|J|$ is generated where each gene value is assigned a random number between 1 and $maxP$ from 4.1.

Job Sequencing: The second portion of the chromosome to be initialized is the precedence preference that determines the job order. The initialization of this portion of the chromosome is particularly complex due to how it interacts with the continuity constraint in the dynamic

decoding. As discussed earlier, to handle the continuity constraint, a machine must be reserved after it has been assigned to the first job in a continuity pair. It can not be assigned to anything else until it is assigned to the second job in the constraint. If there is a situation where the number of jobs with the continuity constraints is larger than the number of robots in the workforce and the jobs all fall within a similar window of the project, it is possible for all the robots to be reserved for different portions of different jobs, making it impossible for jobs in-between the constrained pairs to be completed. This leads to job precedence chromosome configurations that are semi-infeasible. They are not strictly infeasible since the precedence constraints are not violated, however, the probability of the machines being allocated in such away that they they will get stuck in a reserved state, preventing the project from proceeding, is very high.

To resolve this, the initialization has two different methods. The first method, represented in pseudocode in Algorithm 1, takes in the list of jobs and the precedence information. It places all the jobs into a not assigned array (NA) and then shuffles them. Then, as long as NA is not empty, all of the jobs remaining have their precedence constraints checked (line 7). If they are met, they are placed in an array (A) to be added to the chromosome (line 10). Completing **this** for loop (line 6 - 12) extracts all the jobs that currently have their precedence constraints met. They are shuffled, removed from NA, and have their IDs added to the chromosome (lines 13 - 16). This is repeated until all of the jobs have been added to the chromosome, at which time the precedence chromosome has been generated.

This algorithm allows for randomness in job ordering between each level of precedence. However, as described above, it can lead to the semi-infeasible condition at the point of dynamic decoding. To address this, Algorithm 2 was developed to include the additional continuity information $\mathcal{C}$. It initializes the same as the previous algorithm. However, starting at line 10, the job is checked to see if it is present in the continuity constraints. If it is, the

Algorithm 1 Precedence chromosome initialization method 1

Require: $J, prec$
1: $chromosome \leftarrow [\,]$
2: $NA \leftarrow J$
3: $shuffle(NA)$
4: **while** $|NA| > 0$ **do**
5: $A \leftarrow [\,]$
6: **for** $j \in NA$ **do**
7: **if** $prec(j) = False$ **then**
8: **continue**
9: **else**
10: $A \leftarrow j$
11: **end if**
12: **end for**
13: $shuffle(A)$
14: **for** $j \in A$ **do**
15: $del\ NA[j]$
16: $chromosome \leftarrow ID(j)$
17: **end for**
18: **end while**
19: **return** $chromosome$

array for adding, A, is shuffled and added to the chromosome like before (lines 11-14). Then the job and its continuity pair (j, j') are added to the chromosome (lines 16 - 24). The rest of the algorithm proceeds the same as the previous one.

This addition places the two jobs in a precedence constraint right next to each other, meaning that right after the machines are put on hold for the first job the next job they will be assigned to through the dynamic decoding is the second job in the constraint, fulfilling the reservation.

This second method, while it does resolve the precedence problem, it does not allow any jobs to be assigned between the two jobs (by design) which may not reflect what the optimal solution could look like. To preserve the ability to insert jobs between continuity constrained jobs when possible and increase the diversity in the initial population as much as possible, the algorithms are called based on some probability. The method in Algorithm 1 is called with 90% chance and Algorithm 2 is called the other 10% of the time. This ensures that, if the project is highly complex, an initial precedence sequence chromosome portion can still be found.

Machine Allocation: Similar to the initialization of the processing plan chromosome portion, the machine allocation chromosome is initialized to be the max possible number of operations that could occur in the project: $\sum_{j \in J} |O_j|$. The value for each gene can take any number between 1 and $maxM$ found from 4.2.

4.3.5 Parent Selection

After the initial population has been generated, the population needs to propagate into future generations through crossover and mutation. Before this can be done, parents have to be selected and paired. Parent selection is a crucial process in the genetic algorithm since it

Algorithm 2 Precedence chromosome initialization method 2

Require: $J, prec, \mathcal{C}$
1: $chromosome \leftarrow [\,]$
2: $NA \leftarrow J$
3: $shuffle(NA)$
4: **while** $|NA| > 0$ **do**
5: $A \leftarrow [\,]$
6: **for** $j \in NA$ **do**
7: **if** $prec(j) = False$ **then**
8: **continue**
9: **else**
10: **if** $j \in \mathcal{C}$ **then**
11: $shuffle(A)$
12: **for** $k \in A$ **do**
13: $del\ NA[k]$
14: $chromosome \leftarrow ID(k)$
15: **end for**
16: $A \leftarrow [\,]$
17: $(j, j') \in \mathcal{C}$
18: $A \leftarrow [j, j']$
19: **for** $k \in A$ **do**
20: $del\ NA[k]$
21: $chromosome \leftarrow ID(k)$
22: **end for**
23: $A \leftarrow [\,]$
24: **else**
25: $A \leftarrow j$
26: **end if**
27: **end if**
28: **end for**
29: $shuffle(A)$
30: **for** $j \in A$ **do**
31: $del\ NA[j]$
32: $chromosome \leftarrow ID(j)$
33: **end for**
34: **end while**
35: **return** *chromosome*

is necessary to maintain the diversity to converge to a good solution [60, 98]. Two possible different selection types are random selection and fitness based selection. While random selection is self-explanatory, fitness based selection methods can take several different forms. These include Roulette Wheel, Stochastic Universal Sampling, and Tournament Selection [36, 71, 74, 124]. As the name suggests, Roulette Wheel selection divides a wheel into n number of portions where n is the number of chromosomes that can be parents. Each portion is weighted by its fitness value, giving the higher fitness chromosomes a higher chance of being selected. Stochastic Universal Sampling is very similar to the Roulette Wheel method. However, it has the added feature that both parents are selected using two different points around the wheel, meaning they are both chosen at once, making it more likely that at least one high fitness parent will be chosen within a pair. The Tournament Selection process functions a little differently. A subset of chromosomes are pulled from the population. The highest fitness parent out of the subset is selected as a parent. This process is repeated until enough parents have been selected.

The method used in this work closely resembles the Tournament method. For each pair of parents, a small subset of chromosomes are chosen. The elite chromosome out of the set for a given parent is chosen with some probability $P_{elitism}$. This process is repeated for each parent selection within a pair until all the required pairs have been selected.

4.3.6 Crossover

The crossover portion of the genetic algorithm is what allows the two parents to produce children to help populate the next generation. There are several different methods to accomplish the crossover that include multiple crossover points such as partially mapped crossover [127], crossover methods based on the order such as ordered crossover [109], single point

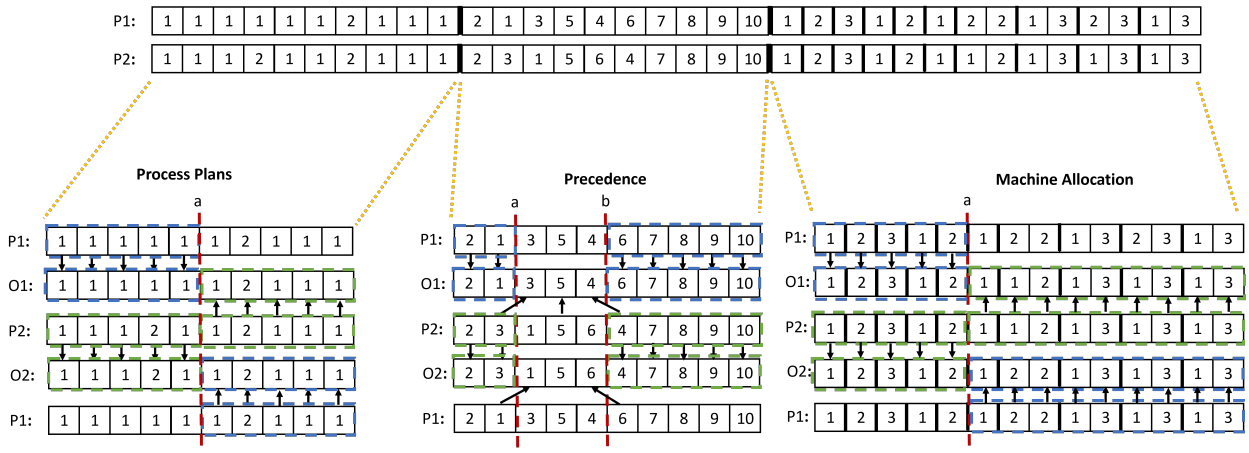


Figure 4.6: The chromosome portions for the process plans and machine allocation both have single point crossover operations. The precedence chromosome portion has a double crossover point to preserve precedence.

crossover methods that rearrange the order [133], and cyclic based crossover [70] to name a few. Each of these have different characteristics as a broad group that are modified to fit specific problem instances.

Since each of the three chromosome portions encode information differently, they are treated separately in the crossover process. Once the parents have been selected, a crossover will occur with some high probability $P_{crossover}$. This allows for some instances where the parents are passed on as the children instead of producing two new child chromosomes from the parents. The crossover method for each of the chromosome parts is represented in Figure 4.6 and explained in the following sections.

Process Plans: The process plan chromosome portion uses a single point crossover. Pictured in Figure 4.6, this method splits how the parent chromosomes go into the children chromosomes based on a randomly chosen point. The blue represent the contributions from the first parent and the green represent the contributions from the second parent. In this example, offspring 1's first portion comes from parent 1 and the second half comes from

parent 2. The reverse is true for offspring 2.

Precedence: Unlike the process plan portion, the precedence chromosome portion requires a double crossover point to preserve the precedence ordering present in the parent chromosomes. The example in Figure 4.6 highlights how this process works. For each offspring, the outside portions of two randomly selected crossover points are copied into the chromosomes. In this case, parent 1 populates offspring 1 and parent 2 populates offspring 2. Next, the parent that did not contribute to the offspring already is read from left to right. If a given value for that chromosome is already accounted for (since no jobs can be repeated in the precedence chromosome) it is passed over. If a value not yet present is found it is placed in the portion between the two crossover points, filling from left to right.

Machine Allocation: The machine allocation chromosome portion utilizes a single point crossover point. Like the process described for the process plan portion, each parent contributes the genes from a different side of the crossover point. This ensures that each offspring contains part of each parent.

4.3.7 Mutation

The mutation portion of the genetic algorithm can be thought of similar to a local area search. It will slightly modify existing chromosomes rather than changing a large portion of the chromosome like the crossover method does. There are several different methods for mutation in the literature that include bit flipping [90] and swapping gene entries [103]. Both swapping and a form of bit flipping are used in this work. Like the crossover steps, the mutation process is also broken into the different chromosome parts. The probability of a chromosome mutating is given by P_{mutate}. How each chromosome is mutated is shown in

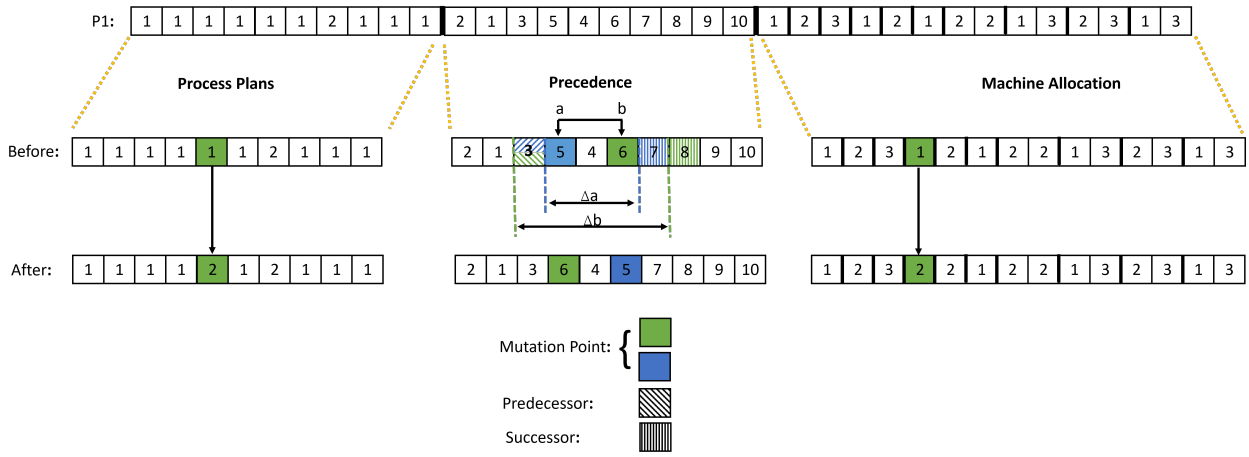


Figure 4.7: Both the process plan and machine allocation are mutated with a form of bit flipping. The precedence portion uses a constrained variant of bit swapping.

Figure 4.7.

Process Plan: To mutate the process plan chromosome, a bit is selected at random. That bit is then randomly assigned a value between 1 and $maxP$. Each value is given a uniform chance of being assigned.

Precedence: Since there can only be one of each value in the precedence chromosome, a swapping method has to be used instead of a bit flip method. One option for this procedure is to allow two randomly selected jobs to be swapped. However, this will lead to an infeasible solution a large percentage of the time for some projects with tight precedence constraints. To address this, a method modeled after a mutation process in [92] is used where the valid range of each selected gene is evaluated. To accomplish this, two genes are selected at random. For each of these genes the predecessor (the last job that must come before it in the precedence constraint) and the successor (the first job that depends on it as a precedence constraint) are found and the range between the two are marked. The swap can only take place if both the genes selected are inside both of the ranges given by the predecessors and successors.

Figure 4.7 demonstrates this with the diagonal and vertical lines color coded (green and blue respectively) according to two randomly selected genes. If this range constraint is met precedence will not be violated and the two positions are swapped. If not, the chromosomes are left alone and passed through unchanged.

Machine Allocation: Since the machine allocation does not have any limitation with repeat values (a result of the dynamic decoding), it can also be mutated with a bit flip mutation. A gene is selected at random with uniform chance and given a value between 1 and $maxM$.

4.3.8 Survivor Selection

In this work, a new set of children (through crossover and mutation as the probabilities allow) are generated until they number the population count of the previous generation. At this point, a survivor selection process has to take place to determine which individuals will be preserved to make up the current (new) generation. There are several ways that this can be done. One process is to use fitness based selection through something like tournament selection described previously. A second possible method is age based selection where the oldest members of the generation are replaced.

The method used in this work uses a form of fitness base selection that also includes a percent elitism feature ($E_s\%$). The top $E_s\%$ of the population (rounded up) with the highest fitness from the older generation are preserved. This allows for the best solutions from the previous generation to remain even if all the new chromosomes are better to help improve the diversity. After this step, the child chromosomes are randomly paired with chromosomes from the previous generation. The chromosome with the highest fitness value in each pair is retained for the next generation. The random pairing allows for some of the lower fitness

valued chromosomes to remain as long as they are paired with a worse new chromosome.

4.3.9 Fitness Evaluation

Adequately modeling the fitness value can be one of the more complex elements to model in a genetic algorithm since it must adequately provide a way for the chromosomes to be ranked against each other. For this particular problem type, the criteria is the makespan. Since the objective of the optimization process is to find a solution that completes the project as close to optimal in the temporal domain as possible, this results in minimizing the makespan. The smaller the makespan, the better the fitness.

The process of solving for the makespan from the decoded solution can be broken down into solving for how long each robot takes to travel between jobs and how long it takes each robot to complete each job. A pseudocode representation of the process used to generate these values is presented in Algorithm 3. The inputs to the fitness calculation requires six inputs. The first is the job order, J_{order}, from the decoded precedence chromosome portion. The next input is the precedence information for each job, $Prec$. Following this is the robot work lists, RW, decoded from the machine allocation information. Next are the processing times, how long it takes a robot to complete each operation, pt, (this comes from the machine ability information). The final two inputs are the distances between all the points, d, taken from the problem definition, and the velocity of each robot, vel, which are also based on the machine ability information.

For each job in the job order (line 2) an evaluation needs to be done to determine the job start and finish times (JT^s and JT^f respectively), the robot travel start time (RT^{Ts}), travel finish time (RT^{Tf}), work start time (RT^{Ws}), and work finish time (RT^{Wf}). These will be recorded for each job and are counted as running tallies to place when in the project time

Algorithm 3 Fitness Evaluation

Require: $J_{order}, Prec, RW, pt, d, vel$

Initialize: $JT \leftarrow [s, f], RT \leftarrow [Ts, Tf, Ws, Wf]$
for $j \in J_{order}$ **do**
  for $r \in RW_j$ **do**
    $RT_j^{W\Delta} \leftarrow pt_{r,o_j} + \frac{d_w}{vel_r}$
    if $RW[j] = 1st$ **then**
      $RT_j^{Ts} \leftarrow 0$
      $RT_j^{Tf} \leftarrow RT_j^{Ts} + \frac{d_t}{vel_r}$
    else
      $RT_j^{Ts} \leftarrow RT_{j-1}^{Wf}$
      $RT_j^{Tf} \leftarrow RT_j^{Ts} + \frac{d_t}{vel_r}$
    end if
  end for
  if $Prec_j > 0$ **then**
    $mJT^f \leftarrow \mathbf{max}(\mathbf{JT_j^f} \in \mathbf{Prec})$
  end if
  if $Prec_j > 0$ && $\mathbf{max(RT_j^{Tf})} < \mathbf{mJT^f}$ **then**
    $JT_j^s \leftarrow mJT^f$
    $JT_j^f \leftarrow JT_j^s + \mathbf{max(RT_j^W)}$
    for $r \in RW_j$ **do**
      $RT_j^{Ws} \leftarrow mJT^f$
      $RT_j^{Wf} \leftarrow RT_j^{Ws} + \mathbf{max(RT_j^W)}$
    end for
  else
    $JT_j^s \leftarrow \mathbf{max(RT_j^{Tf})}$
    $JT_j^f \leftarrow JT_j^s + \mathbf{max(RT_j^W)}$
    for $r \in RW_j$ **do**
      $RT_j^{Ws} \leftarrow \mathbf{max(RT_j^{Tf})}$
      $RT_j^{Wf} \leftarrow RT_j^{Ws} + \mathbf{max(RT_j^W)}$
    end for
  end if
end for
$makespan \leftarrow JT_{last}^f$
return $makespan, RT, JT$

span jobs were completed and which robots were working on them. First, the amount of work time is solved for based on the processing time for that robot on the operation it has been assigned ($pt_{r,oj}$) and the distance it has to travel during the operation divided by the robot's velocity capability ($\frac{d_w}{vel_r}$). A check is done to see if the job is the first for that robot (lines 5-11). If it is, the travel time RT_j^{Ts} is set to 0 and the finishing time of travel is set to the start time plus the amount of distance it has to travel to get to the job divided by the robots velocity ($\frac{d_t}{vel_r}$). If it is not the first job the travel start time is based off when the robot finished the last job it was working on (RT_{j-1}^{Wf}). The ending time of travel is once again when it started plus the time it took to travel the required distance.

After work time has been calculated for each robot for job j, a check is done to see if job has any precedence constraints (lines 13-15). This check is necessary since a job can not be started until its precedence constraints are met. This means that if there are precedence constraints and if the last robot working on the job has arrived before the precedence jobs were completed (line 16), the start time of the job will occur when the last precedence job was completed (mJT^f). This also affects when that job was completed, JT_j^f, (this is shown on lines 17-18). The start (RT_j^{Ws}) and finish (RT_j^{Wf}) work times of the robots on that job are also based on this information (lines 19-21). It is important to note that the start and finish work times are the same for each robot, based on the longest work time / the slowest robot on the job, ($\mathbf{max}(\mathbf{RT_j^W})$). This is due to the fact that the operations represent a robot contribution and if there are multiple operations they are dependent on the other robots contribution, otherwise they would be two different jobs.

If the job j does not have any precedence requirements or if they are all completed before the last robot arrives, a similar process takes place (lines 23-30). However, the start time of the job is just based on when the last robot arrives, $\mathbf{max}(\mathbf{RT_j^{Tf}})$. After these steps have been completed for every job, the makespan can be defined as the completion time of the last job

(line 32). The makespan along with the robot times RT and job times JT are returned out of the fitness algorithm.

4.4 Experiments

To evaluate the genetic algorithm formulation developed here, a set of experiments have been designed to test how the current formulation responds to different parameter configures. In addition to this sensitivity test, the resulting schedules from these trials will be compared to the optimization process preformed by the MIP presented in Chapter 3.

4.4.1 Parameter Sensitivity

The first set of experiments seek to evaluate this formulation's sensitivity to the parameters. It is a common practice to compare a new genetic algorithm formulation against other formulations with known problems to see if the new formulation improves the solving time or the optimized result. However, as discussed earlier in this chapter, this formulation was developed *because* no formulations could be found that could handle all the constraints present in this problem, specifically the continuity constraints. As such, the evaluation of this formulation will take the form of running on a few different types of assembly problems seen in this work.

Arch Assembly: The first project set used in the sensitivity analysis is the arch assembly developed in Chapter 3. To evaluate scalability, the arch assembly is solved with a single arch, as done in Chapter 3, as well as a two arch and a three arch case. For each of these cases, each arch has the same internal constraints and there are no precedence constraints

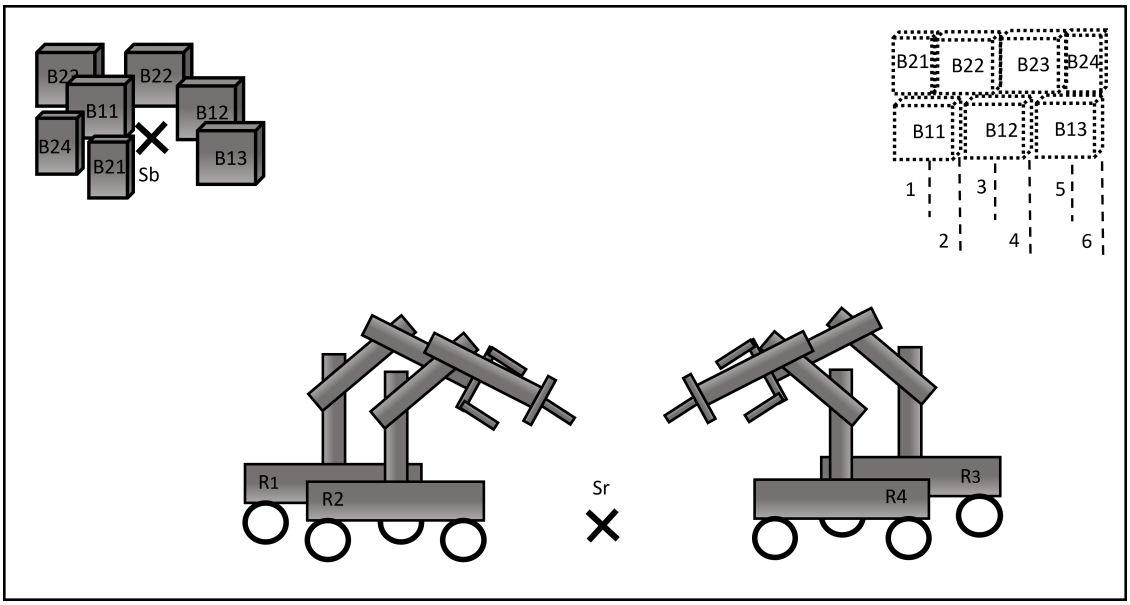


Figure 4.8: Workspace for truss block project for the three base block by two rows instance.

between arches. That is to say, the workforce can start work on all the arches at the same time or choose to complete one fully before going on to the next arch.

Truss Wall Assembly: The second type of assembly is a truss wall assembly. In this project, two teams of robots are tasked with building a truss block wall. Pictured in Figure 4.8, this project has more complexity in the constraints as shown in Figures B.1 and B.2 in Appendix B. Like the arch project, the truss wall was scaled to several different levels to give a range of complexity and to evaluate how well the genetic algorithm handles the different size projects. The SAPD for this assembly is given in Appendix B.

Parameter Ranges: Three of the parameters expected to have the most impact on the solution convergence time and optimality were varied for the sensitivity analysis. The first is the crossover probability, $P_{crossover}$. This probability directly affected the search capability of the genetic algorithm since it represents the chance of parent chromosomes producing two

offspring. If this probability is low, there is a lower rate of search through combinations of working schedules. In this analysis, the the crossover probability was evaluated over the set: $P_{crossover} \in \{0.3, 0.6, 0.9\}$

The second parameter that was varied in the sensitivity analysis was the mutation probability, P_{mutate}. Like the first parameter, this probability also impacted the exploration of the solution space since it directly affected how solutions were perturbed to search the local area around the solutions. In this analysis, the mutation probability was varied across the set: $P_{mutate} = \{0.4, 0.6, 0.8, 1.0\}$.

In addition to the crossover and mutation probabilities, the initial population was also varied in this sensitivity analysis. While the other two parameters control how the formulation navigates through the search space, the initial population impacts the number of solutions that are searched each generation. The larger the starting population, the wider the search begins in the case of this particular formulation. However, this has a trade off. Since this formulation preserves the number of the population through the generations, the larger the initial population, the longer it will take each generation cycle. For this analysis, the initial population was varied across the set: $IP \in \{50, 150, 250, 500\}$.

The remaining parameters that were held constant were the elite keep percentage at 5%, the probability that a pair would successfully breed (95%), and the probability that the more elite parent would be chosen out of the two pairs of two for each breeding (90%). Additionally, each run was set to a max of 500 generations with the ability for early stopping. If the best generations did not have a change in fitness value over 20 generations, the model would consider the solution converged and move on to the next combination in the sensitivity analysis. A combination would also be cut off it the run took longer than 12 hours. Due to the stochastic nature of the genetic algorithm combined with the stochastic nature of the projects, each combination was run 10 times to find the average makespan and the average

solving time of each combination.

4.4.2 Results

The results presented here were run on a Ubuntu 20.04 server with 128 GB of RAM and a 3.7 GHz 32 core CPU. The genetic algorithm analysis did not run in parallel and the mixed integer program results in this work were limited to 40 threads. The mixed integer program was also limited to 12 hours per run due to other tasks that required the server.

Sensitivity Analysis: The best performing parameter configuration was selected for each project by first selecting the best makespan (averaged across the 10 runs). If there were multiple configurations that tied for a given project, the one with the faster convergence time was selected. The results from this analysis are given in Table 4.1. In addition to the parameters, the table shows the statistical information concerning the makespan, solving time, and the number of generations needed to find the best solution (recall that if the makespan did not improve for 20 runs the simulation considered that run converged). The results indicated that the ideal crossover and mutation probabilities vary depending on the project. In contrast, the largest initial population was also preferred to generate the best results. This can be attributed to an increased exploration capability of the GA due to a larger population each iteration.

As the projects scale in size, here measured by the number of jobs, the solving time also increased. Figure 4.9 shows another representation of the best performing parameter configurations, picturing the magnitude of change across the two projects as they scale. The error noted on the graphs is the standard deviation of that value across the samples for the project and parameter configuration. In Panel A of Figure 4.9, the makespan increases in an approximately linear manner for each of the two project types which is to be expected.

Table 4.1: Best genetic algorithm parameter configuration for each project.

$P_{elitism} = 0.05$			$P_{parent\ elitism} = 0.9$				$P_{successful\ breeding} = 0.95$		
$Max\ Generation = 500$			$Samples\ per\ Combination = 10$						
Project	$P_{crossover}$	P_{mutate}	Init. Pop.	$\mu_{makespan}(sec)$	$\sigma^2_{makespan}$	$\mu_{SolveTime}(sec)$	$\sigma^2_{SolveTime}$	$\mu_{Generation}$	$\sigma^2_{Generation}$
Arch Assembly	3.00E-01	8.00E-01	5.00E+01	6.99E+02	3.62E+03	2.92E-02	2.55E-04	3.12E+01	2.96E+02
2 Arch Assembly	6.00E-01	1.00E+00	5.00E+02	1.12E+03	1.02E+04	1.18E+00	1.19E-01	7.13E+01	4.17E+02
3 Arch Assembly	9.00E-01	8.00E-01	5.00E+02	1.50E+03	5.76E+04	2.02E+00	3.90E-01	8.00E+01	6.23E+02
Scaling wall 5 x 2	3.00E-01	1.00E+00	5.00E+02	2.43E+02	1.64E+02	1.25E+00	1.78E-01	5.91E+01	4.13E+02
Scaling wall 5 x 3	9.00E-01	1.00E+00	5.00E+02	3.36E+02	3.25E+02	2.86E+00	3.17E-01	8.54E+01	2.90E+02
Scaling wall 10 x 2	9.00E-01	6.00E-01	5.00E+02	5.10E+02	5.32E+02	4.19E+00	1.02E+00	9.80E+01	5.68E+02
Scaling wall 10 x 3	9.00E-01	8.00E-01	5.00E+02	7.07E+02	4.74E+02	8.70E+00	4.12E+00	1.26E+02	8.66E+02
Scaling wall 20 x 3	6.00E-01	1.00E+00	5.00E+02	1.75E+03	2.14E+03	3.11E+01	1.22E+01	2.05E+02	5.34E+02

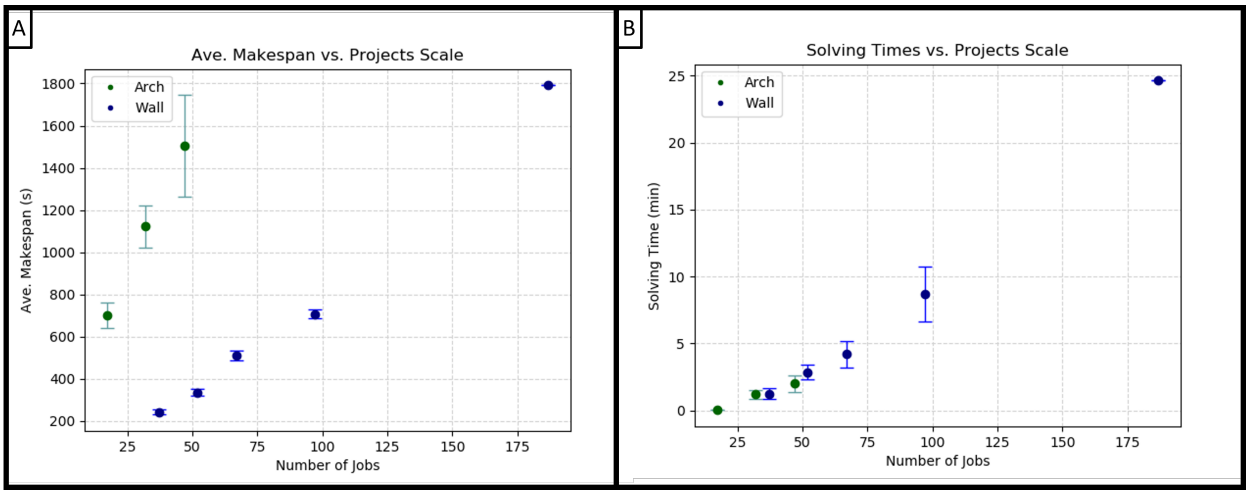


Figure 4.9: Panel A: The average makespan vs. the number of jobs for the best parameter configuration per project. Panel B: The average solving time vs. the number of jobs for the best parameter configuration per project.

The solving time, given in Panel B, begins to take on an exponential shape as the projects scale, increasing in both the solving time and variation in how long it takes to solve.

Comparison to MIP: Running the MIP formulation on the same projects resulted in information shown in Table 4.2. It is important to note that, with the 12 hrs limit, the two largest project did not have an initial solution found. Here, the single arch assembly was able to converge to an optimal solution (a gap of 0%). The remaining arch projects and the smaller wall projects were not able to converge, resulting in gaps ranging between 17% to 76%. The two largest truss wall assemblies were not able to solve for a valid solution in the allotted time. This is represented by a -1 value in the makespan column and a gap of infinite.

Using these results from the MIP, a quantifiable estimated for how close the GA solutions are to an optimal solution can be found. In the case where the MIP converged to a provable optimal (a gap of 0%), the MIP makespan was used as a reference. In the case where the MIP did not converge, the lower bound of the MIP was used. While this makespan does

Table 4.2: MIP results for the arch and truss wall projects.

Project	Makespan	Lower Bound	Gap	Time Optimizing (min)
Arch Assembly	6.79E+02	7.30E+02	0.00E+00	5.85E-02
2 Arch Assembly	1.22E+03	6.01E+02	5.71E-01	7.20E+02
3 Arch Assembly	1.83E+03	5.46E+02	7.56E-01	7.20E+02
Scaling wall 5 x 2	2.05E+02	2.01E+02	1.67E-01	7.20E+02
Scaling wall 5 x 3	2.90E+02	2.19E+02	3.96E-01	7.20E+02
Scaling wall 10 x 2	4.15E+02	3.20E+02	4.16E-01	7.20E+02
Scaling wall 10 x 3	-1.00E+00	3.85E+02	INF	7.20E+02
Scaling wall 20 x 3	-1.00E+00	9.58E+02	INF	7.20E+02

not come from a viable solution to the entire MIP, it provides a worse case measurement since the optimal solution makespan is somewhere between it and the current GA solution. This measurement, in seconds, is given in the "Dist. Opt." column of Table 4.3. The "% Diff. Makespan" gives the percent difference between the best MIP solution and the best GA solution. The final three columns provide the solving times for both models and the percent difference between them.

A graphical comparison between the makespan that the mixed integer program returned and the average makespan found by the genetic algorithm are shown in Figure 4.10. As mentioned before, the two largest projects could not solve for a valid makespan. These are the two -1 entries on Panel B.

To further evaluate the two scheduling methods, 10,000 processing time samples were taken for each robot / operation pair utilized in each of the projects (Appendix C.2 includes examples of these distributions for the single arch and 5x2 truss wall projects). Using these sampled times, a makespan distribution was generated using a best parameter configuration schedule from the GA and the best MIP schedule. Figure 4.11 shows a comparison of the distributions for the three truss wall projects both methods solved. The remaining two truss wall projects that only the GA could solve in the time limit are shown in Figure C.1 in

Table 4.3: Comparison between the GA solutions and the MIP solutions for each project.

Project	GA Makespan (s)	MIP Makespan (s)	MIP LB (s)	Dist. Opt.	% Diff. Makespan	MIP Gap	GA Time (min)	MIP Time (min)	% Diff. Time
Arch Assembly	7.47E+02	6.79E+02	7.30E+02	6.83E+01	2.40E-02	0.00E+00	2.62E-02	5.85E-02	1.90E-01
2 Arch Assembly	1.19E+03	1.22E+03	6.01E+02	5.88E+02	1.64E-01	5.71E-01	9.10E-01	7.20E+02	4.99E-01
3 Arch Assembly	1.36E+03	1.83E+03	5.46E+02	8.16E+02	2.14E-01	7.56E-01	1.29E+00	7.20E+02	4.98E-01
Scaling wall 5 x 2	2.42E+02	2.05E+02	2.01E+02	4.17E+01	4.70E-02	1.67E-01	1.53E+00	7.20E+02	4.98E-01
Scaling wall 5 x 3	3.26E+02	2.90E+02	2.19E+02	1.07E+02	9.80E-02	3.96E-01	2.88E+00	7.20E+02	4.96E-01
Scaling wall 10 x 2	5.18E+02	4.15E+02	3.20E+02	1.98E+02	1.18E-01	4.16E-01	3.30E+00	7.20E+02	4.95E-01
Scaling wall 10 x 3	6.99E+02	-1.00E+00	3.85E+02	3.14E+02	1.45E-01	INF	7.47E+00	7.20E+02	4.90E-01
Scaling wall 20 x 3	1.74E+03	-1.00E+00	9.58E+02	7.84E+02	1.45E-01	INF	3.29E+01	7.20E+02	4.56E-01

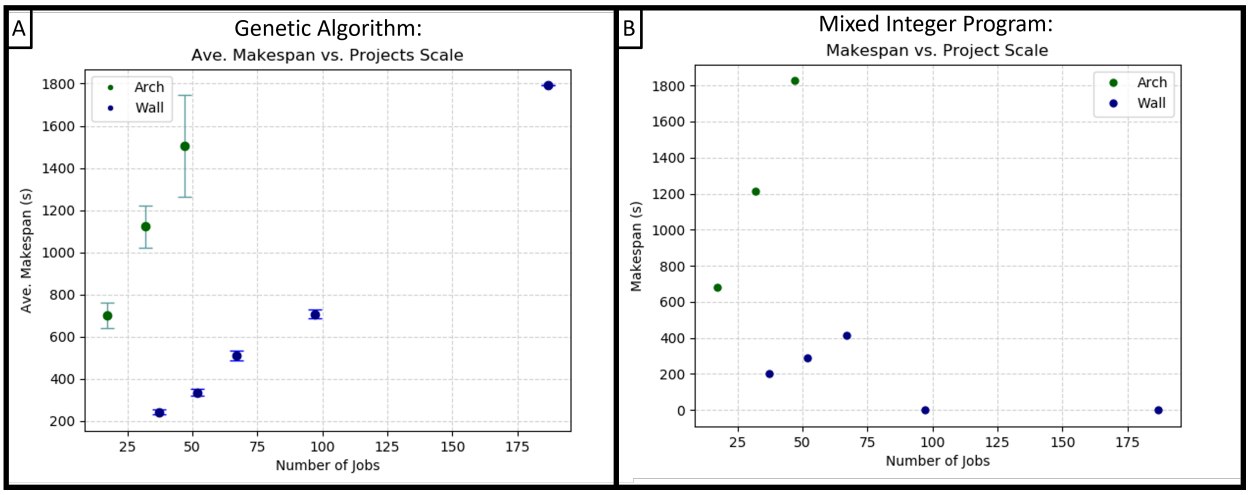


Figure 4.10: Panel A: Plot of makespan vs. project scale for genetic algorithm. Panel B: Plot of makespan vs. project scale for mixed integer programming.

Appendix E.

In contrast to the truss wall assembly projects, the arch assembly projects did not have clean distributions. Due to the nature of how the processing times stacks for the outliers sample from the processing times, the makespan distributions contain some extreme outliers. This is a result of the stochastic information for this project set being generated from the teleoperation information as discussed in Chapter 3. Figure 4.12 shows the results for the single arch case where Panel A contains the full distribution with the vast majority grouped into a single bin with the outliers pushing the mean much higher. To provide a clearer picture of this distribution it is broken into the majority distribution, shown in Panel B, and the outlier distribution, shown in Panel C. Similar distributions for the double and triple arch problems are shown in Figure C.2 in Appendix E. A summary of the different means for the arch assembly case are given in Table 4.4.

As a final comparison used the mean values of the processing times distributions. The results from this can be seen in Figure 4.13.

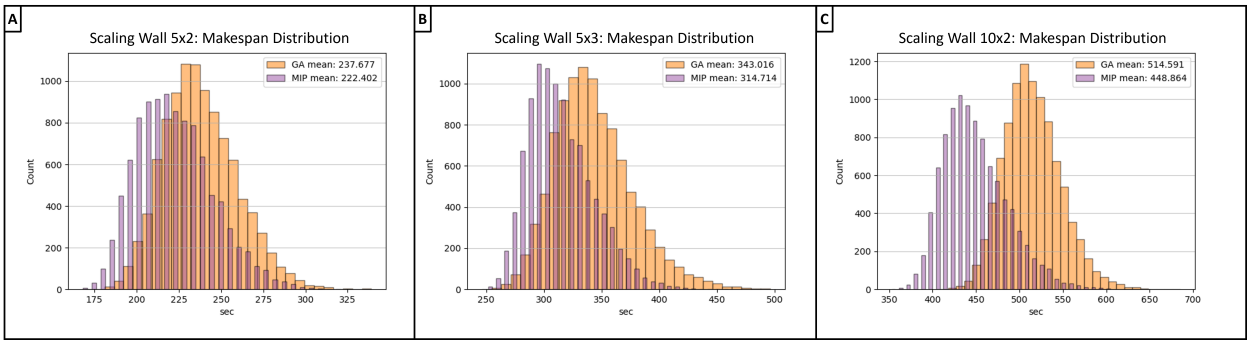


Figure 4.11: Panel A: The 5 x 2 truss wall makespan distributions generated from 10,000 processing time instances. Panel B: The 5 x 3 truss wall makespan distributions generated from 10,000 processing time instances. Panel C: The 10 x 2 truss wall makespan distributions generated from 10,000 processing time instances.

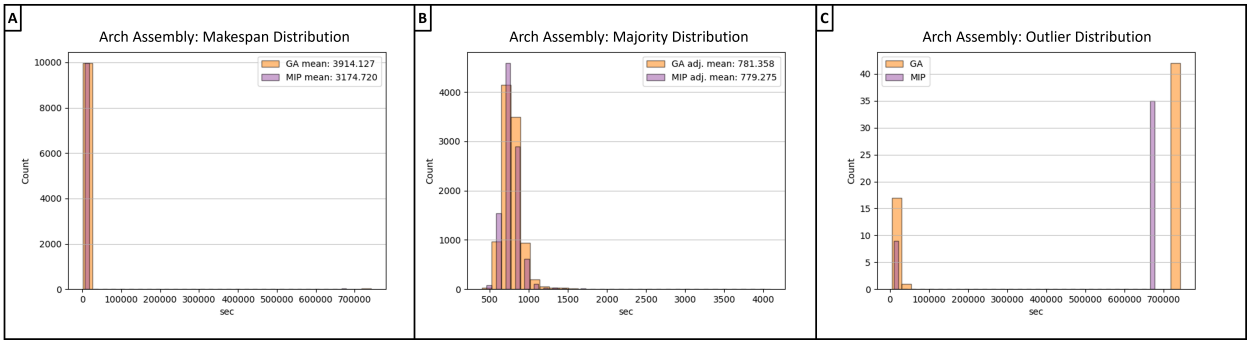


Figure 4.12: Panel A: Single arch full makespan distributions generated from 10,000 processing time instances. Panel B: Single arch majority makespan distributions generated from 10,000 processing time instances. Panel C: Single arch outlier makespan distributions generated from 10,000 processing time instances.

Table 4.4: Arch assembly project makespan distribution means and adjusted means (means of non-outlier points).

Project	Scheduler	Mean (sec)	Adj. Mean (sec)
Single Arch Assembly	GA	3914.13	781.36
	MIP	3174.72	779.28
Double Arch Assembly	GA	7249.48	1627.06
	MIP	8270.56	2410.76
Triple Arch Assembly	GA	8270.56	2410.76
	MIP	10139.40	2056.61

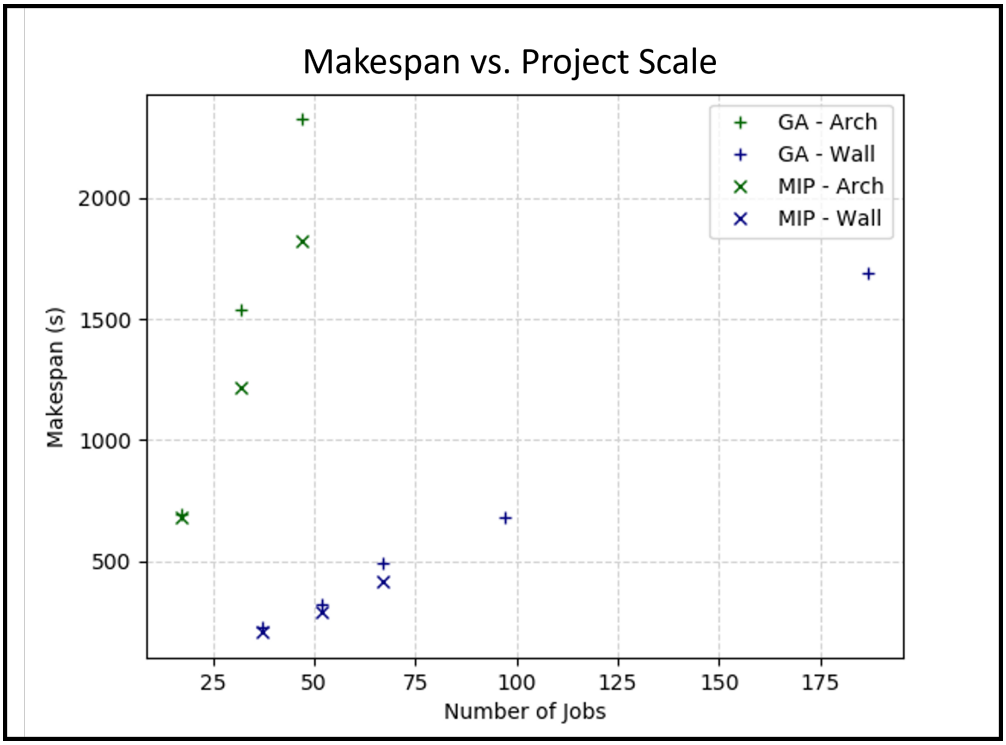


Figure 4.13: The makespan (calculated using mean processing times) vs. number of jobs for each scheduling method and assembly project.

4.4.3 Discussion

The sensitivity analysis for the genetic algorithm indicates that, for this formulation, maximizing the search capability by increasing the solution variation and increasing the number of times this occurs through each generation (the population number) provided the best results. This makes intuitive sense given that the goal is to find the best makespan as quickly as possible. If computation concerns are a consideration, a different minimization can be done to find the parameters that produced the best results with the least amount of computation (or up to an allowable limit). This would primarily impact the initial population parameter.

When comparing with the MIP results, it can be seen that not all of the MIP formulations converged to an optimal. Looking the percent difference in the makespan between the two methods, the genetic algorithm was able to get within a 2% to 21% difference from the optimal worst case (the lower bound of the MIP) depending on the project. Ideally, the MIP would be allowed to run until it converged or until it could close the gap for some of the larger projects, allowing for a better estimate for the worst case measurement from optimality for the GA. Additionally, future work can look at adding to the MIP constraint set to help the solver carve up the solution space.

4.5 Summary of Contribution

The work in this chapter adds to the work in this dissertation that addresses the second research question inquiring about solving for an optimal or near optimal solution to the task allocation and sequencing problem. The formulation developed here extends the concepts present in existing literature to handle the dynamic nature of the constraints necessary for

the in-space robotic assembly problem.

While the work in Chapter 3 provides a method for generating an optimized schedule that can be quantitatively described in terms of how close to the optimal it is, this genetic algorithm formulation provides a method that utilizes the stochastic information in a greater capacity. By repetitively sampling from the processing time distributions, this formulation is more capable of handling more complex distribution shapes since it is not reliant on the mean of the distribution.

Chapter 5

Ability Analysis

In the context of task sequencing and allocation, an important element to consider is how well a robot can complete a given task. Robot capability will have large impact on the allocation process in the search for an optimal assembly schedule. As such, it is helpful to develop a criteria of capabilities that should be checked or approximated to inform the scheduling problem when the robot can not experimentally complete a task type before the assembly.

If a robot can not reach each point (pose) required in a task or if it can not bear the loads seen in the task it is not a machine that can be assigned to that task and still retain a feasible solution. Similarly, how well a machine can complete a task can impact both the allocation and sequencing. All else being equal, the scheduler may need to change the sequence of the tasks to find an optimal solution, allowing for certain machines to work on certain tasks based on how long it will take the machines to complete the tasks. As such, having an estimate of if a robot can reach the required poses in a task, if it can handle the load requirements, and how long it will take to complete the tasks are necessary information.

It is also helpful to have some estimate of chance of failure if that information is going to be including in the scheduling process. For in-space assembly specifically, there may be situations when failure estimation ability will be important. Situations may occur in which ability information is needed for a robot and can not be experimentally estimated (an example of this would be if a robot unexpectedly broke when deployed and a different robot

had to be tasked with the first robots work).

5.1 Research Gap

In the literature, [115] provides three criteria consisting of processing time, additional investment, and process quality. Other proposed metrics, like those in [89, 132], include criteria such as reachability and weight as metrics to quantify ability. Other scheduling research such as [106] categorize the criteria as repeatability, efficiency, accuracy, and load bearing. [83] frames the capabilities as dexterity (focused on reachability) and fatigue, focused on torque / motor capability. In addition to payload and reach, speed and gripper type have also been suggested for the criteria [96].

Reachability is often categorized as workspace analysis in the literature. Defining the workspace will often take the form of kinematically checking the workspace, sampling it for valid poses [12, 45, 49, 135]. When trying to define the whole workspace as a pre-calculated map, the analysis is commonly referred to as a capability map, where each position has many orientations sampled from it [46, 95, 147]. The ratio of reachable orientations out of the total evaluated at a specific position can give a score quantifying how dexterous the robot is at that point in the workspace [146]. Searching this workspace is very computationally expensive. As such, methods such as Monte Carlo sampling have been used to mitigate this [59, 87]. To provide a measure for how close to a singularity a specific pose is, an analysis of the manipulability ellipsoid can be completed to estimate how close the Jacobian is to a singularity for a given configuration [134]. From a processing time perspective for the purpose of autonomous assembly the estimation of how long it will take a robot to do something does not have a lot of literature representation. In [76] robot speed for certain tasks was estimated based on how long it will take a robot to traverse a distance. Along this line

of thought, estimations for how long it would take a robot end effector to traverse a path (moving between points) which can be calculated using kinematics [94, 125]. Estimating the load bearing capability of a robot can be done through a kinematic static [13, 48] or dynamic analysis [47, 139, 148]. To analyze the loads along a path the inverse models can be used [81].

5.2 Ability Categories

One method for gathering ability information is through experimental trial as seen in the hardware experiments in Chapter 3. However, as stated earlier, in the context of a deployed autonomous assembly team, it is possible tasks will arise that can not be experimentally tested ahead of time. As such, having an analytical way to approximate the ability of a robot to complete specific tasks is an important capability that directly ties into the allocation and sequencing process. This chapter will propose four criteria to be used to analyze a robot's ability to complete a task for this allocation paradigm. In addition to the criteria, an example of how the criteria might be approximated will be developed and discussed.

For the purpose of quantifying the ability of a robot to complete a task in the context of autonomous assembly, four different categories are defined. The first is **reachability**. This category will cover if a robot is capable of reaching all of the necessary points in the workspace to complete the task. The analysis for this category should be able to determine if a robot has a valid pose to each of the points in the task and if a path is defined, the points along that path would also need to be analyzed. The second category is **processing time**. This category will include an analysis to quantify an estimate for how long the robot will take to complete a given task. This will include the time it takes the robot to complete all of the steps to move across all of the required points and the use of whatever tool or

end effector is in use. The third category will be a **capability** analysis. In this portion, a method for approximating torques applied to the motors on an arm will be discussed. This type of analysis will target determining if any of the motors will not be able to support the load applied to the end effector during the completion of a task. Finally, the **probability of failure** category will be developed. This category will provide a framework to factor in uncertainties that are present in the task process to approximate how likely a robot is to fail.

In this work, a formulation will be proposed to demonstrate the base use of these criteria and provide an estimation for a system with limited modeling requirements (i.e. a lack of the full inertia information, dynamic models of the system, and project specific failure considerations). While including additional considerations in the implementation of the metrics is outside the scope of what is presented here, it will be clear how more complex modeling methods can be brought in to provide better estimates for the proposed criteria.

5.3 Kinematic Formulation

To approximate if a robot is capable of reaching a task, estimating how long it will take to move, or if it can bear a load, a model must be used to approximate how the robot moves. This is a primary goal of a kinematic representation of a robot. It is able to model where points of robot (such as the end effector) are in space, what their orientation is, and how these points can be moved using the joints of the robot as inputs. Since this model is important in the context of ability analysis, this section will provide some of the background theory in the kinematics using sources such as [30, 94, 125, 137]. As this chapter develops, it will become apparent how this kinematic formulation is used to inform the proposed criteria.

5.3.1 Spatial Transformation

A spatial point can be broken down into two main elements: *orientation* and *position*. As pictured in Figure 5.1, orientation can be thought of as a description of how the basis vectors of one coordinate frame are expressed in terms of the basis vectors of a reference frame. This can be thought of as a dot product between basis vectors in the two coordinate frames since the dot product of two until vectors is the cosine of the angle between them. Using this definition, a matrix can be defined representing this difference in orientation, the rotation:

$$\mathbf{R}_{sb} = \begin{bmatrix} \widehat{\mathbf{x}}_\mathbf{b} \cdot \widehat{\mathbf{x}}_\mathbf{s} & \widehat{\mathbf{y}}_\mathbf{b} \cdot \widehat{\mathbf{x}}_\mathbf{s} & \widehat{\mathbf{z}}_\mathbf{b} \cdot \widehat{\mathbf{x}}_\mathbf{s} \\ \widehat{\mathbf{x}}_\mathbf{b} \cdot \widehat{\mathbf{y}}_\mathbf{s} & \widehat{\mathbf{y}}_\mathbf{b} \cdot \widehat{\mathbf{y}}_\mathbf{s} & \widehat{\mathbf{z}}_\mathbf{b} \cdot \widehat{\mathbf{y}}_\mathbf{s} \\ \widehat{\mathbf{x}}_\mathbf{b} \cdot \widehat{\mathbf{z}}_\mathbf{s} & \widehat{\mathbf{y}}_\mathbf{b} \cdot \widehat{\mathbf{z}}_\mathbf{s} & \widehat{\mathbf{z}}_\mathbf{b} \cdot \widehat{\mathbf{z}}_\mathbf{s} \end{bmatrix} \tag{5.1}$$

where R_{sb} represents the coordinate frame defined by $\widehat{\mathbf{x}}_\mathbf{b}, \widehat{\mathbf{y}}_\mathbf{b}, \widehat{\mathbf{z}}_\mathbf{b}$ in terms of the frame defined by $\widehat{\mathbf{x}}_\mathbf{s}, \widehat{\mathbf{y}}_\mathbf{s}, \widehat{\mathbf{z}}_\mathbf{s}$.

To represent an orientation in $\mathcal{R}^3$, the rotation matrix can be split into three rotational steps representing how a coordinate frame is represented with respect to a reference by the amount of rotation around the z, y, and x axes respectively. This is the Euler Angle [94] representation of orientation where (ϕ, ψ, θ) represent the rotations about (z, y, x). Using the rotation of these angles to solve for their respective rotation matrices, 5.1 can then be defined as the combination of all three rotations:

$$\mathbf{R} = \mathbf{R}_\mathbf{z}(\phi)\mathbf{R}_\mathbf{x}(\theta)\mathbf{R}_\mathbf{y}(\psi). \tag{5.2}$$

To describe the translation difference between two points or reference frames a simple vector can be used. The spatial displacement vector between the two frames can be defined as:

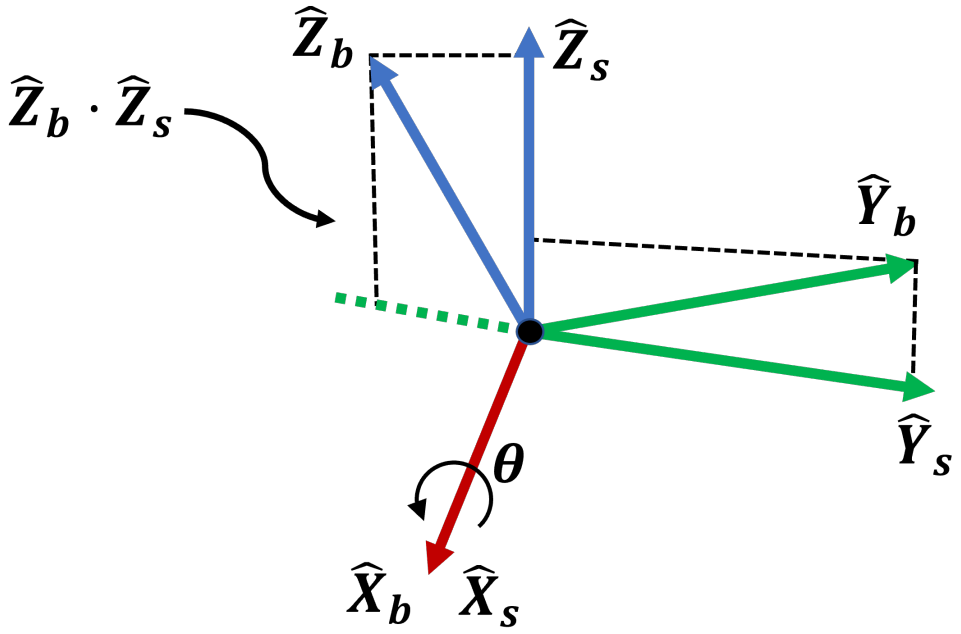


Figure 5.1: Example of the dot product representation of the rotation between two sets of unit basis vectors representing two coordinate frames.

$$\mathbf{p} = [\mathbf{x}, \mathbf{y}, \mathbf{z}]^{\mathbf{T}} \tag{5.3}$$

Using both the rotational matrix solved for in 5.2 and the position vector in 5.3, a $4x4$ spatial transform can then be defined to fully describe a coordinate frame in $\mathcal{R}^3$:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix} \tag{5.4}$$

A point of interested in the assembly problem, point a, can be defined with respect to the workspace reference frame, s. The position and orientation of point a is then defined by T_{sa}. If a description is needed to map the spatial frame with respect to the point, the transform can be inverted, solving for the position and orientation of s with respect to a: $T_{sa} = T_{as}^{-1}$.

5.3.2 Screw Theory

While the Euler Angle representation of orientation combined with a spatial displacement vector is an effective way to describe points of interest within the workspace, an alternative way of representing this information is preferred for the kinematic representation of the robots. This work uses the screw theory formulation over the Danavit-Hartenberg parameter [40] configuration for the global description of the rigid body motion [141] of the robots in the assembly workforce.

Exponential Coordinates of Rotation: For the purpose of screw theory, a rotation is framed in terms of a rotation axis, $\widehat{\mathbf{w}}$, by some amount of rotation, θ. This framing of rotation is placed into the exponential form using Rodrigues' formula derived with a Taylor series expansion [94]:

$$\mathbf{R} = \mathbf{e}^{[\widehat{\mathbf{w}}]\theta} = \mathbf{I} + \mathbf{sin}\theta[\widehat{\mathbf{w}}] + (\mathbf{1} - \mathbf{cos}\theta)[\widehat{\mathbf{w}}]^{\mathbf{2}} \tag{5.5}$$

where $[\cdot]$ is the skew-symmetric matrix used to represent a cross product in matrix multiplication [94]. Given $a = [a_1, a_2, a_3]^T$ the skew is defined as [94]:

$$[a] = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \tag{5.6}$$

For the purpose of screw theory, a rotational motion around the axes of rotation can be thought of as an angular rotation around the unit vector: $\mathbf{w} = \widehat{\mathbf{w}}\,\dot{\theta}$. The rate of rotation for each axis in the reference frame can then be expressed in matrix form as: $\dot{\mathbf{R}} = \mathbf{w} \times \mathbf{R}$ or,

using the skew representation:

$$\dot{\mathbf{R}} = [\mathbf{w}]\mathbf{R} \tag{5.7}$$

In a similar way, the change in the transformation (rotation and position) can be thought of as $\dot{\mathbf{T}} = [\mathcal{V}]\mathbf{T}$ where $\mathcal{V}$, defined as the twist, represents the angular and tangential velocities:

$$\mathcal{V} = \begin{bmatrix} w \\ v \end{bmatrix} \tag{5.8}$$

With a stretch of notation, $\mathcal{V}$ can be presented in a "skew" matrix form:

$$[\mathcal{V}] = \begin{bmatrix} [w] & v \\ 0 & 0 \end{bmatrix} \tag{5.9}$$

This twist can be defined in terms of the workspace spatial frame $\{s\}$ by post-multiplying the change in the transform by the inverse of the transform: $[\mathcal{V}_s] = \dot{T}T^{-1}$ or in terms of the body frame (the frame of the object moving) by pre-multiplying: $[\mathcal{V}_b] = T^{-1}\dot{T}$. The body frame representation provides the most straightforward understanding of the twist. In $\mathcal{V}_b = [w_b\ v_b]^T$, w_b is the angular velocity of the object in the body frame $\{b\}$ and v_b is the linear velocity of the object described with respect to the body frame. In the space frame $\{s\}$ variant of the twist, w_s is the angular velocity of the object in terms of $\{s\}$ and v_s mathematically works out to be the instantaneous velocity of a point on the body of the object at the origin of $\{s\}$ *if* the object is imagined to be stretched such that is spans the origin of $\{s\}$.

To convert the twist between the two frames, an adjoint mapping is used [94] to map based

on transform T:

$$[Ad_T] = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ [\mathbf{p}]\mathbf{R} & \mathbf{R} \end{bmatrix} \tag{5.10}$$

where R is R_{sb} and p is the position of $\{b\}$ with respect to $\{s\}$. This mapping allows for easy conversion between the two reference frames: $\mathcal{V}_s = Ad_T(\mathcal{V}_b)$. This adjoint mapping will be important later on in the ability analysis.

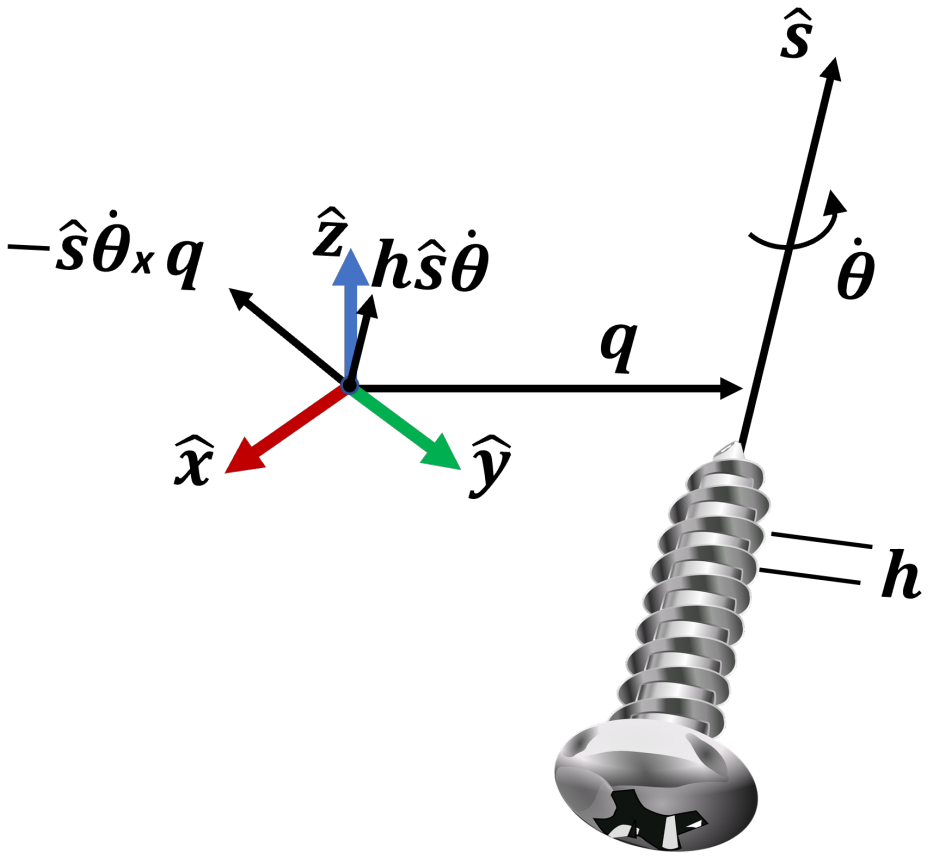


Figure 5.2: A screw axis $\mathcal{S}$ represented by a point defined by the vector q from the reference frame in the unit direction $\hat{s}$ with a pitch h.

Recall that the angular velocity can be presented as a rate of rotation $\dot{\theta}$ about some axis $\hat{\mathbf{w}}$. In a similar way, the $\mathcal{V}$ can be thought of as some rate of rotation $\dot{\theta}$ about a **screw axis** $\mathcal{S}$. As the name indicates, this can be thought of in terms of a screw. Pictured in Figure

5.2, $\mathcal{V}$ represents the motion along the threads of a screw meaning that as it rotates about the axis it also translates along the axis in a manner determined by the thread pitch. The angular portion of velocity is just the rotation about the axis: $w = \hat{s}\dot{\theta}$. The linear velocity is a combination of the translational components: $h\hat{s}\dot{\theta}$ where h is the thread pitch, and the linear motion at the origin of the reference frame: $-\hat{s}\dot{\theta} \times q$ where q is the vector from the reference frame to the screw axis. As such, the $\mathcal{V}$ can be formally defined as:

$$\mathcal{V} = \begin{bmatrix} w \\ v \end{bmatrix} = \begin{bmatrix} \hat{s}\dot{\theta} \\ -\hat{s}\dot{\theta} \times q + h\hat{s}\dot{\theta} \end{bmatrix} \tag{5.11}$$

noting that the screw axis, $\mathcal{S}$, is defined by some set $\{q, \hat{s}, h\}$. To simplify what is required to define the screw, two cases can be considered:

1. If there is no angular velocity present ($w = 0$) then the pitch of the screw is infinite. In this case, $\hat{s}$ is chosen as $v/||v||$ to capture the motion due only to the linear velocity. In this case $\mathcal{V} = \mathcal{S}\dot{\theta}$ with $\dot{\theta}$ representing the magnitude of the linear velocity: $\dot{\theta} = ||v||$.

2. If there is an angular velocity present ($w \neq 0$), the screw axis is defined such that $\hat{s} = w/||w||$ with $\dot{\theta} = ||w||$, $h = \hat{w}^T v/\dot{\theta}$, and q is chosen such that linear motion is orthogonal to $\hat{s}$. This leads to: $\mathcal{V} = \mathcal{S}\dot{\theta}$ with $\dot{\theta} = ||w||$.

Keeping these two cases in mind, the normalized screw axis is defined as:

$$\mathcal{S} = \begin{bmatrix} w \\ v \end{bmatrix} \tag{5.12}$$

where either (i) $w = ||1||$ and $v = -w \times q + hw$ or (ii) $w = 0$ and $||v|| = 1$. Condition (i) can be used to model purely rotational motion by setting the pitch to zero ($h = 0$). Conversely, condition (ii) can be used to model purely translational motion. Many robotic joints can

be broken down into producing either rotational or translational motion, which will be built upon shortly.

Remembering the exponential representation of rotation about an axis, a similar form can be derived with Taylor series expansion utilizing $\mathcal{S}$ as the axis. This results in the form [94]:

$$\mathbf{T} = e^{[\mathcal{S}]\theta} = \begin{bmatrix} e^{[\mathbf{w}]\theta} & (I\theta + (1 - cos\theta)[\mathbf{w}] + (\theta - \mathbf{sin}\theta)[\mathbf{w}]^{\mathbf{2}})\mathbf{v} \\ \mathbf{0} & 1 \end{bmatrix} \tag{5.13}$$

where $e^{[\mathbf{w}]\theta}$ is defined in Equation 5.5. This exponential representation of an orientation and position in space is instrumental in the kinematic modeling of a robotic arm. A kinematic formulation for a robotic arm often consists of a product of homogeneous transformation matrices that describe points of interest on the robot. For the exponential form, this is accomplished using the product of exponentials formula. In this formulation, the position and orientation of the end effector frame on the robot is defined when each joint is in its zero position noted by M. This is considered the "home" position of the robot. Each joint of relevance, those leading up to the point of interested are defined by a screw. If the joint is a revolute joint, the screw axis is defined according to condition (i). If it is prismatic the screw axis is defined according to condition (ii). Using this definition philosophy, a transformation chain can be defined using the product of exponentials to describe how M will be modified based on each joint's motion:

$$T(\theta) = \mathbf{e}^{[\mathcal{S}_\mathbf{1}]\theta_\mathbf{1}} \dots \mathbf{e}^{[\mathcal{S}_\mathbf{n-1}]\theta_\mathbf{n-1}} \mathbf{e}^{[\mathcal{S}_\mathbf{n}]\theta_\mathbf{n}} \mathbf{M} \tag{5.14}$$

where $(\theta_1 \dots \theta_n)$ and $(\mathcal{S}_1 \dots \mathcal{S}_n)$ are the joint orientations and screw axes for each joint leading up to M, respectively. Using the example pictured in Figure 5.3, M would be defined as:

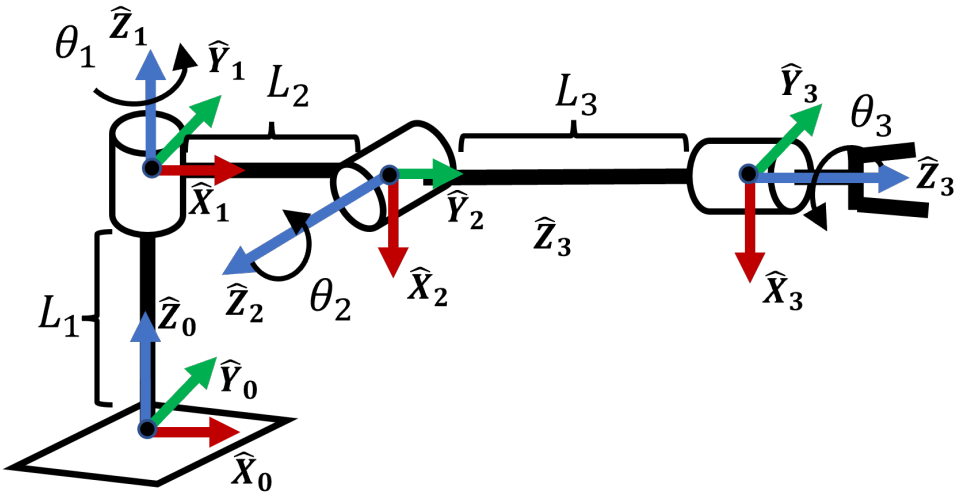


Figure 5.3: Schematic of a three degrees of freedom robotic arm.

Table 5.1: Screw definitions for robot in Figure 5.3.

i	w_i	v_i
1	$(0,0,1)$	$(0,0,0)$
2	$(0,-1,0)$	$(L_1,0,-L_2)$
3	$(1,0,0)$	$(0,L_1,0)$

$$M = \begin{bmatrix} 0 & 0 & 1 & L2+L3 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & L1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and the screws would be defined as shown in Table 5.1 resulting in the following skew screw representations:

$$[\mathcal{S}_1] = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$[\mathcal{S}_2] = \begin{bmatrix} 0 & 0 & -1 & L_1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -L_2 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$[\mathcal{S}_3] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & L_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Putting this together, the forward kinematic representation of the arm can be represented with the product of exponential form:

$$T(\theta) = e^{[\mathcal{S}_1]\theta_1} e^{[\mathcal{S}_2]\theta_2} e^{[\mathcal{S}_3]\theta_3} M$$

5.3.3 Inverse Kinematics

The forward kinematic formulation, presented in 5.14, provides a way to solve for the end effector location and orientation given each of the joint angles. The inverse kinematic for-

mulation provides a way to reverse this process. Given a desired end effector location and orientation, the inverse kinematics seeks to solve for a valid joint configuration that will place the end effector in the desired spatial position and orientation. Inverse kinematics be can be solved analytically or numerically. The analytical solution is often difficult to solve (for serial robot arms) and may need to be done piecewise with geometrically defined equations. In addition to solving for a valid joint configuration, it will also provide a way for the hardware joint bounds of the robot to be considered in the solution analysis.

The process of solving for the joint angles starts with the development of the Jacobian, J, for a given robot. The Jacobian used in this work is the geometric Jacobian, which can be thought of as a mapping between the joint velocities, $\dot{\theta}$, and the end effector, $\mathcal{V}$. In contrast to the geometric Jacobian, the analytical Jacobian (not used here) provides a mapping between the joint velocities and time derivative of the end effector coordinates in the spatial frame $\{s\}$. Since the inverse kinematic method used here will be based around the twist, the Jacobian will be of the geometric type.

As discussed in 5.3.2, $\dot{T} = [\mathcal{V}]T$, which can be rearranged to solve for the twist of the end effector with respect to the spatial coordinate frame s: $[\mathcal{V}_s] = \dot{T}T^{-1}$. This can be manipulated into the form:

$$\mathcal{V}_s = J_s(\theta)\dot{\theta} \tag{5.15}$$

where J_s is the spatial Jacobian matrix: $[J_{s1}, J_{s2}(\theta), \ldots, J_{sn}(\theta)]$ where $J_{si}(\theta)$ is the adjoint mapping (Equation 5.10) of the transformation leading up to screw i. As an example, for $i = 3$, $J_{s3}(\theta) = Ad_{e^{[\mathcal{S}_1]\theta_1}e^{[\mathcal{S}_2]\theta_2}}(\mathcal{S}_3)$.

To test if a point has a valid inverse kinematic solution, a twist can be solved for that will encapsulate the required motion to move the robot end effector from its home position M

into the correct location and orientation. First, a transform, $T_{transfer}$, is defined as the difference between the point transform T_p and the home end effector location M:

$$T_{transfer} = T_p M^{-1} \tag{5.16}$$

This comes from the fact that M is the transform of the end effector with respect to the spatial frame and T_p is the transform of where the end effector needs to go with respect to the spatial frame.

This transform is then used to solve for the twist required to move the end effector to that location using the matrix log form [94]:

$$\boldsymbol{\nu}_s = [\mathcal{S}]\theta = \begin{bmatrix} [w]\theta & v\theta \\ 0 & 0 \end{bmatrix} \tag{5.17}$$

Having found $\boldsymbol{\nu}_s$, Equation 5.15 can be rearranged using the inverse (or the Moore-Penrose inverse as a pseudoinverse to approximate the inverse of the Jacobian for non-square or singularity cases [125]) to solve for the required change in the joints:

$$\dot{\theta} = J_s^{\dagger} \boldsymbol{\nu}_s \tag{5.18}$$

Using this formulation directly will weight each joint identically based on the two norm. This is not ideal in that, in this form, the kinematics does not know the physical bounds of each joint. For example, if an elbow joint of a robot is limited between $[30^o, 100^o]$ the kinematics may still solve for a solution that will require the elbow to be 120^o. As such, the joints need to be weighted according to how close the current solution is to a bound. This is accomplished by adding in the null space of the Jacobian. Since the Jacobian is the

mapping between the joints and the end effector, this null space contribution to the solution will not impact the end effector location, just the configuration of the arm as it attempts to reach that desired location. This method, based on the Geometric Projection Method (GPM) [30, 125, 137] takes the form:

$$\dot{\theta} = J_s^{\dagger}\boldsymbol{\nu}_s + (I - J_s^{\dagger}J_s)\mathbf{w} \tag{5.19}$$

Where w is the weighting for each joint. As discussed, for this workspace criteria check, the weights need to be based on how close the joints are to their respective joint bounds. To accomplish this, the cost function:

$$H(\theta) = \frac{1}{2}\sum_{i=1}^{N}\left(\frac{\theta_i - \theta_{i,mid}}{\theta_{i,max} - \theta_{i,min}}\right)^2 \tag{5.20}$$

is defined [125] where the minimum and maximum bounds for joint i are defined as $[\theta_{i,min}, \theta_{i,max}]$ and the midpoint in the range is: $\theta_{i,mid} = \frac{\theta_{i,max} - \theta_{i,min}}{2} + \theta_{i,min}$. The gradient of this cost function provides a weighting for each joint:

$$\nabla H(\theta) = \frac{\partial H(\theta)}{\partial \theta_i} = \frac{\theta_i - \theta_{i,mid}}{(\theta_{i,max} - \theta_{i,min})^2} \tag{5.21}$$

Combining these equations, the form to solve for the joint velocities based on the desired twist and weight in the Jacobian null space according to a joint position gradient becomes:

$$\dot{\theta} = J_s^{\dagger}\boldsymbol{\nu}_s - k(I - J_s^{\dagger}J_s)\nabla H(\theta) \tag{5.22}$$

where θ is the current configuration of the robot and k is a scalar value to weight the impact of the null space solution. To numerically solve for a joint configuration, the joint velocity

is split across a unit time step: $\dot{\theta} = \theta^{t+1} - \theta^t$ to solve with an Euler integration:

$$\theta^{t+1} = \theta^t + J_s^\dagger \mathcal{V}_s - k(I - J_s^\dagger J_s)\nabla H(\theta^t) \tag{5.23}$$

As shown in Algorithm 4, the process of solving for the joint values θ takes the initial position (or a guess) set of joint values, solves for transformation describing the motion and then the twist (lines 4 and 5 respectively). Until the bounds are met and the iterative $\mathcal{V}$ is small enough to indicate the required motion is within the error tolerance or the max number of iterations has reached the algorithm keeps updating θ (lines 10 and 11).

Algorithm 4 Inverse Kinematics Algorithm

Require: $\mathcal{S}, T_{goal}, M, \theta_g, bounds, k, err, iter$
1: $good = false$
2: **for** $i = 1, i++, i < iter$ **do**
3: $\quad T_g = e^{[\mathcal{S}_1]\theta_{g1}} \dots e^{[\mathcal{S}_{n-1}]\theta_{gn-1}} e^{[\mathcal{S}_n]\theta_{gn}} M$
4: $\quad T_{tf} = T_{goal} T_g^{-1}$
5: $\quad \mathcal{V}_s = \mathbf{TMtoTwist}(\mathbf{T_{tf}})$
6: $\quad$ **if** $\mathcal{V}_s \leq err$ && **checkbounds**(θ_g) **then**
7: $\quad\quad good = true$
8: $\quad\quad$ **break**
9: $\quad$ **end if**
10: $\quad J_s = [J_{screws}(\theta_g)]$
11: $\quad \theta_g = \theta_g + J_s^\dagger \mathcal{V}_s - k(I - J_s^\dagger J_s)\nabla H(\theta_g)$
12: **end for**
13: **return** $\theta_g, good$

5.3.4 Wrench

The Jacobian mapping capability also has a use in the static analysis of load bearing capabilities of a robot. This work will look a static analysis method through the kinematics previously defined. To begin this analysis, the load at the end of a robot must be defined.

This is done by defining a six element representation of the moment and force, the wrench:

$$\mathcal{F} = \begin{bmatrix} m \\ f \end{bmatrix} \in \mathcal{R}^6 \tag{5.24}$$

Where m the three element moment and f is the three element load force. As long as these can be calculated for the end effector at a given point (for example, the weight of an object being held by the robot) and the robot is not in motion, a static analysis can be considered. Since this is a static analysis, the principle of conservation of power can be applied to note that the power at the joints equals the power to move the robot plus the power required for the load at the end effector. Here, the power to move the robot is zero allowing the power at the joints to be equated to just the power required for the load on the end effector. This leads to the equation:

$$\tau = J^T(\theta)\mathcal{F} \tag{5.25}$$

where τ is the vector of the joint torques. If the load is being applied with some offset from the last frame of reference, it can be transformed to a new reference frame using the adjoint of the transform describing the difference between the reference frames:

$$\mathcal{F}_b = [Ad_{T_{ab}}]^T \mathcal{F}_a \tag{5.26}$$

Using these formulations, the torques required to hold an object at a pose can be calculated. When the masses of the robot links are not small with respect to the load being carried, it is desirable to include them (as weights in this case) for the robot links in the load estimation. To accomplish this, the torques resulting from each link, being treated as a payload for the

remaining portion of the arm, can be calculated separately and then linearly combined with each other and the torques due to the object being carried.

5.4 Ability Categories Developed

The kinematic formulation provides the base theory that will be used as examples on how to analyze a robot with respect to the four ability analysis categories.

5.4.1 Reachability

As discussed earlier, the reachability category of the ability analysis quantifies if portions of the points within a task are reachable for the different robots on the workforce. In this work, this analysis consists of using inverse kinematic to determine if each point contains valid inverse kinematic solution.

If we let P_{task} represent the set of points required for a given task, each point is evaluated for a given robot's kinematics. If each point in the set has a valid inverse kinematic solution, then that task passes the reachability criteria of the ability analysis for that robot. This process takes the following form show in Algorithm 5:

Algorithm 5 Inverse Kinematics Algorithm

Require: $\mathcal{S}, M, bounds, P_{task}$
1: **for** $p \in P_{task}$ **do**
2: $T_p \leftarrow p$
3: $\theta, status = \textbf{IKalg}(\mathcal{S}, \mathbf{T_p}, \mathbf{M}, \dots)$
4: **if** $status = false$ **then**
5: **return** $false$
6: **end if**
7: **end for**
8: **return** $true$

The inputs to the function are the robot parameters such as the screws, the home frame, joint bounds, and the set of points required for the task. For each point, the inverse kinematics algorithm described in Algorithm 4 seeks a valid configuration (line 2). If any of the points return as invalid, the reachability is returned as false. If all of the points are reachable, the analysis returns true noting that the reachability metric has passed.

5.4.2 Processing Time

The amount of time a robot will take to complete a task can also be estimated using the kinematics discussed above. For each point i in the list of points in a task $\forall p \in P_{task} \backslash \{p_1\}$ (where p_1 is the starting point), the processing time can be thought of in two parts. The first is the amount of time it takes to move from the previous point to the current point: $t^i_{\Delta p}$ with $\Delta p = p_i \leftarrow p_{i-1}$. The second is the amount of time that must be spent at that point: t^i_p. The total estimate for a task can then be estimated as:

$$PT_{task} = \sum_{i=2}^{|P_{task}|} t^i_{\Delta p} + t^i_p \tag{5.27}$$

where p is the ith point in the set P_{task}. The travel time between the current and previous point can be estimated using the inverse kinematic equation, 5.23, with a slight variation. As a consequence of how the twist is defined, the path a twist will naturally follow is an arc or helical path vs. a straight line. For the purpose of following desired paths with the end effector, it is desireable to be able to calculate a twist that will cause the end effector to move in a straight path with respect to the spatial reference frame $\{s\}$ (see Figure 5.4 for a comparison). To accomplish this, a *decoupled* twist needs to be generated. This decoupling separates out the rotation from the translation.

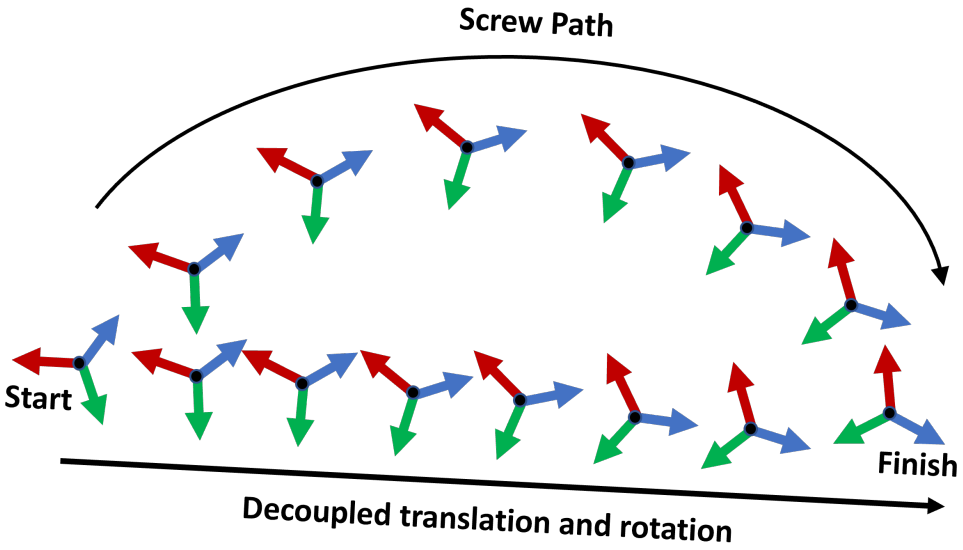


Figure 5.4: The path following constant screw motion verses a path defined by the decoupled representation.

Considering an example where the end effector of a robot needs to be moved from a to b where there is a translation and rotational difference between the two point frames. The decoupled twist, $\mathcal{V}_d$ can be defined as:

$$\mathcal{V}_d = \begin{bmatrix} w_{tr} \\ p_b - p_a \end{bmatrix} \tag{5.28}$$

where w_{tr} is the angular rotation of the twist solved for using Equations 5.16 & 5.17 and p_b and p_a are the translational vectors to points b and a with respect to frame $\{s\}$. While this successfully decouples the translational and rotation motion in the twist, $\mathcal{V}_d$ is with respect to the frame at the starting point of the motion. As such, it needs to be converted to the space frame using the transform:

$$T_{p_{sa}} = \begin{bmatrix} I & p_a \\ \mathbf{0} & 1 \end{bmatrix} \tag{5.29}$$

This transform is used through the adjoint (Equation 5.10) to convert the twist to the proper reference frame: $\mathcal{V}_s = Ad(T_{p_a})\mathcal{V}_d$. It should be noted that this twist must continually be solved for as the robot end effector moves since it will continually change with the end effector's position.

To approximate $t^i_{\Delta p}$ for p_i and p_{i-1} two different parts are considered. First is the path that will be traversed and the second is how a controller will required the robot to move along that trajectory. Depending on the formulation of the controller and trajectory generator, this can be estimated different ways. For the work presented here an s-path representation of a trajectory is going to be used that separates out the geometric description of the path $\theta(s)$ from the temporal aspect $s(t)$. An Euler integration is used to generate $\theta(s)$ (it should be noted that this can be changed to a more accurate variant of numerical integration if needed).

To generate the time independent path, an s-path algorithm is defined in Algorithm 6:

Algorithm 6 S-path Generation

Require: $\theta_0, \mathcal{S}, M, \theta_{min,max}, T_{goal}, N_{steps}$
1: $\delta = \frac{1}{N_{steps}}$
2: $\theta_{path} = [\theta_0]$
3: **for** $i = 2 : N_{steps} + 1$ **do**
4: $\quad T_{ee} = \mathbf{FK}(\theta_{path}[i-1], \mathcal{S}, M)$
5: $\quad J = \mathbf{CalcJ}(\theta_{path}[i-1], \mathcal{S})$
6: $\quad k = \frac{N_{steps}}{N_{steps}-i+2}$
7: $\quad \mathcal{V}_d = k \times \mathbf{DecoupledVels}(T_{ee}, T_{goal})$
8: $\quad \dot{\theta} = J^{\dagger} Ad(T_{p_{ee}}) * \mathcal{V}_d - k(I - J^{\dagger}J)\nabla H(\theta, \theta_{min,max})$
9: $\quad \theta_{path} \leftarrow \theta_{path}[i-1] + \delta * \dot{\theta}$
10: **end for**
11: **return** θ_{path}

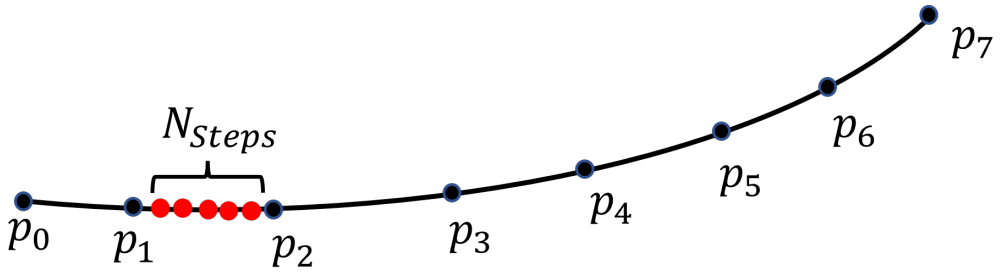


Figure 5.5: Example demonstrating how the points along a path within a task can be discretized to find the processing time.

The input to this algorithm consists of the current joint configuration, θ_0, the screws, home pose, and joint bounds for the robot being evaluated ($\mathcal{S}$, M, and $\theta_{min,max}$), as well as the transform of the desired position and orientation (T_{goal}), and the number of steps to divide the path into, N_{steps} (pictured in Figure 5.5). The decoupled twist is solved for (lines 4-7) based on the decoupling approach discussed above. It is important to note that, due to this being based on a unit time step, a scaling factor needs to be placed to scale the $\mathcal{V}$ (line 7). The returned θ_{path} is $\theta(s)$ where $s[0, \ldots, i, \ldots, 1]$ with i incriminating by $\frac{1}{N_{steps}}$.

To map from the s-space to time, a function needs to be defined to map the s parameter to time a point in time: $s : [0, T] \rightarrow [0, 1]$. Where T is the time to complete the whole trajectory. There are several ways to develop this mapping depending on the controller and velocity profiles. In most cases, where the trajectory is defined between the p and $p-1$, $t^i_{\Delta p} = T$ for the estimate. If the trajectory is used to cross through multiple sets of points then the mapping can be used to pull of the time spent for a given transition between the two positions along the trajectory s, s' corresponding to the points $p-1$ and p on the trajectory.

Next, the time it takes the tool to complete its task at the point needs to be estimated. This value is not as simple to develop a standard estimation method for since it is very dependent on the point in the task and the task itself. For many of the points along a path of motion, this value will simply be zero since the robot's tool does not have a specific process

to complete during the robot motion. For the purpose of formal definition, this time can be defined as:

$$t_p^i = \begin{cases} \textbf{ProcTime}(\text{tool}, p_i), & \text{Operation at } p_i \\ 0, & \text{No operation at } p_i \end{cases} \tag{5.30}$$

where **ProcTime**() changes depending on the type of tool and the operation at p_i in the task. If the tool is something like a gripper that needs to open and close, this can be modeling using the kinematics method describe earlier in this chapter. If it is something like a riveting tool, then it would be a function of how long it takes the tool to place a rivet.

Using this formulation, an estimate can be calculated for how long a task will take for a robot. From a stochastic standpoint, this estimate can be thought of as a mean. If there is variability due to uncertainty, an estimate can be taken for the conditions that would lead to both a faster and longer task time, providing an estimated variance. This variance may take the form of a tool operating slower or adding another point set to the point list to model to account for the robot needing another attempt to successfully arrive at a location.

5.4.3 Capability

In this implementation, the capability being evaluated is the load carrying ability of the robot. There are several different methods that can be used to approximate a robot's ability to carry a load. A coarse approximation along the lines of a static analysis can be completed for the points of interest in the task [94]. Alternatively, a more complex method can include modeling the dynamics of the robot to take into account the dynamic forces due to the masses in motion [125]. This work will consider the simpler analysis for the approximation, defined in Algorithm 7:

Algorithm 7 Estimating Load Bearing Capability

Require: $P, \mathcal{S}, \tau_{max}, M, \theta_P, Object$
1: **for** $p \in P$ **do**
2: $T_{ee} = \mathbf{FK}(\theta_\mathbf{P}, \mathcal{S}, \mathbf{M})$
3: $\mathcal{F} = \mathbf{CalcEELoad}(\mathbf{T_{ee}}, \mathbf{Object})$
4: $\tau = J^T(\theta_P)\mathcal{F}$
5: **if** $\tau > \tau_{max}$ **then**
6: **return** $false$
7: **end if**
8: **end for**
9: **return** $true$

To evaluate load bearing capability, some subset of points (for example, those that will put the robot at an extreme pose) can be defined out of the set of points in a task if the whole set is not going to be analyzed: $P \subseteq P_{task}$. This set of points will then be stepped through using the joint thetas that define the arm configuration at each point $\theta_{p \in P}$, to find the orientation of end effector (T_{ee}). This, along with the properties of the object, can be used to estimate the wrench, $\mathcal{F}$, at the end of the arm. If $\mathcal{F}$ causes the joint torques (τ) to exceed the allowed limits (τ_{max}) the algorithm returns that this criteria has not been met.

5.4.4 Probability of Failure

The final criteria in the ability analysis is the chance of failure. As discussed elsewhere, there are many different factors that can contribute to a robot's chance of failure. These include: the tolerances required in a task vs. the resolution of the sensors and motor control, the grip or hold a robot has on objects it is grasping, how well the robot can maneuver around a workspace, how fragile the objects that the robot is working with are, etc. The scope of this section is not to model each condition but rather to provide a framework for the probabilities to factor into. Generally speaking, the probabilities of failure can be broken down into two general categories: the probability of minor failure, P_{minor}, and the probability of major

failure, P_{major}. Having two different probability categories allows for a distinction between a chance of the robot needing to repeat some portion of the task and the robot causing damage to itself or something it is working on or with. This is an important distinction for a system that will be operating away from human input.

What specifically falls into each category is very project specific. For example, if a robot is carrying a truss object and tries to place it at a specific location on the ground and fails, it may just have to pick back up the object and shift it slightly. In this case, the chance of failure only impacts the processing time. If the chance of failure is high, the processing time can simply include a second (or n number) of attempts in the estimation. However, if this is taking place in a zero gravity environment, it is possible that this slight misplacement can cause the truss object to not be received by retaining clips and result in the object being lost to space. Just by this change of conditions, the failure would fall into the category of a major failure and the allocation system would not want to choose a robot for this task even if there is a relatively small chance of failure.

As an example, this work will look at the probability that the robot end effector will be in a valid location with respect to the goal position given the resolution of the joint inputs. If we let Σ_θ be the diagonal matrix representing the variance of each joint, a covariance matrix for the end effector location can be defined as: $\Sigma_{ee} = J\Sigma_\theta J^T$. For the sake of this example, only the position will be considered since it is more important than the orientation for the following experiments. The geometric Jacobian, a 6×7 matrix for a 7 degree of freedom arm, can be broken into the components that map to the angular velocity (J_w) and the parts that map to the tangential velocity (J_v):

$$J = \begin{bmatrix} J_w \\ \\ J_v \end{bmatrix} \tag{5.31}$$

For the positional consideration, only the tangential component needs to be used: $\Sigma_{ee} = J_v \Sigma_\theta J_v^T$. This Σ_{ee} can then be used to define a multivariate normal distribution that describes the possible end effector locations given the uncertainty in the joint input:

$$\phi(x) = \left(\frac{1}{2\pi}\right)^{3/2} |\Sigma_{ee}|^{-1/2} e^{-\frac{1}{2}(x-\mu)^T \Sigma_{ee}^{-1}(x-\mu)} \tag{5.32}$$

where μ is the goal position for the robot end effector. To utilize the distribution to find the probability that a component will be placed in a correct location, an acceptable volume $V_{tolerance}$ needs to be defined that represents the spatial tolerance for success. Typically, this will be a function of the assembly problem. Integrating over the overlap of the distribution with the tolerance volume provides the probability that the end effector will end up in a valid location:

$$P_{success} = \int_V \phi(x) dV \tag{5.33}$$

where V is $\phi \cap V_{tolerance}$. The probably of failure is then: $P_{failure} = 1 - P_{success}$. In the experiment section a Monte Carlo method will be utilized to estimate this integral.

5.5 Experiments

To demonstrate the use of the ability analysis, an experiment set was performed using an autonomous robot currently being developed in the FASER lab [2]. The experiments below will evaluate reachability, estimate processing time to place a component, provided a static analysis with the heaviest of the payloads, and estimate the probability of the end effector ending up in a valid region.

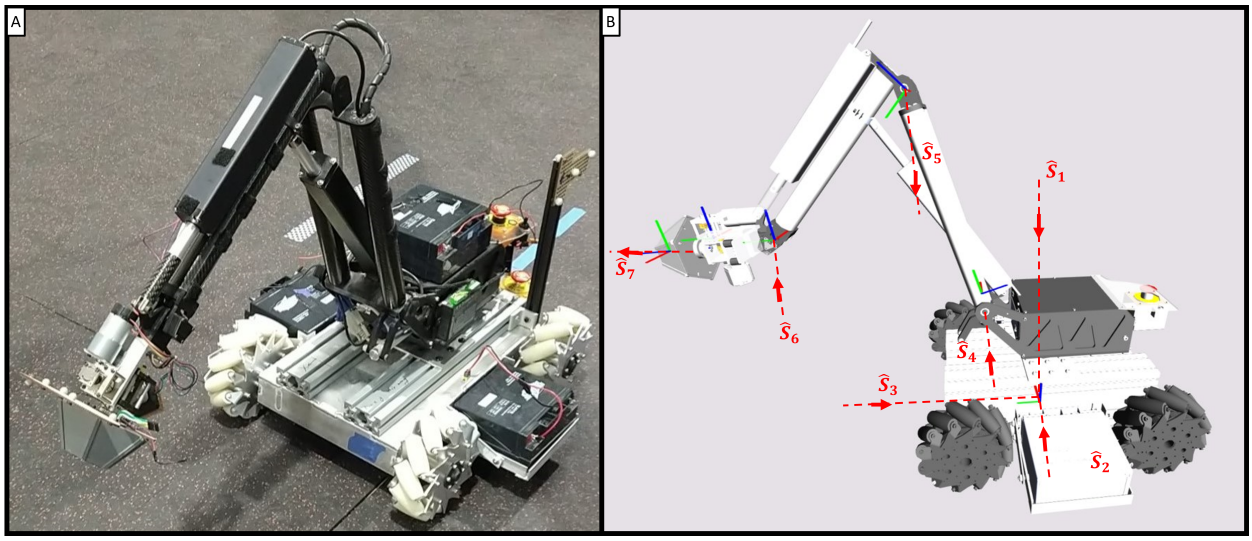


Figure 5.6: Degrees of freedom for the MARC robot.

5.5.1 Hardware

The robot used in this experiment was the Mobile Assembly Robotic Collaborator (MARC) [100]. It consists of an omnidirectional platform (through the use of Mecanum wheels) upon which a robotic arm is mounted. In the following experiments, this system was used as a seven degree of freedom robot where the screws are defined as pictured in Figure 5.6 Panel B. The end effector for this robot is a controllable electro-permanent magnet (EPM) [1] tool that can grab objects with a passive alignment feature.

5.5.2 Autonomy

The autonomy to control the MARC utilizes an OptiTrack [4] camera system to provide positional feedback based on marker sets defined on the robot end effector and platform base. To handle the state estimation an Ensemble Kalman Filter (EnKF) is used where the measurement is the location of the markers on the robot, the states are the joint value estimates, and the control values are the velocities sent out to the joints. The EnKF is a

Bayesian filter derived from the linear Kalman Filter [119] and functions by generating a normally distributed set of points in a Monte Carlo-esque fashion and reporting the mean value and cumulative covariance to aide in state estimation. The formulation of the EnKF used in this work given in Appendix D.

Combined with the metrology system and state estimation is the controller portion of the autonomy. It should be noted that, in the work presented here, the development of a well tuned controller is outside the scope of this dissertation and is still under development for the system. As such, the controller in this work scales according to the joint limits for the hardware (keeping a consistent ratio to preserve the twist path) and will scale down the velocities proportional to the distance remaining after a specified threshold from a goal. The impact of such a controller will be discussed in the experiment results and discussion.

5.5.3 Ability Implementation

For these experiments, the reachability analysis will be completed following Algorithm 5 where each point of interest will be tested for a reachable pose. In a similar manner, the processing time will be approximated using Algorithm 6. Due to the nature of the controller, as discussed above, the robot will operating near the velocity limit for at least one of the joints during the majority of the trajectory. As such, T can be estimated by solving for the joint velocities, $\dot{\theta}$, based on the decoupled twist required to traverse each pair of points. Since th twist is evaluated for a unit measurement of time (seconds in this case), it must then be scaled according to the ratio of the largest difference between the solved velocities in unit time and the max velocities: $T \approx max\left\{\left|\dot{\theta}_{unit\ time} \oslash \dot{\theta}_{max}\right|\right\}$ where none of the $\dot{\theta}_{max}$ are zero. This is the equivalent of using $\Delta t = \frac{T}{N}$ for each time step in the s-path. As discussed in Section 5.4.2, this formulation is specific to the hardware and autonomy at hand. For

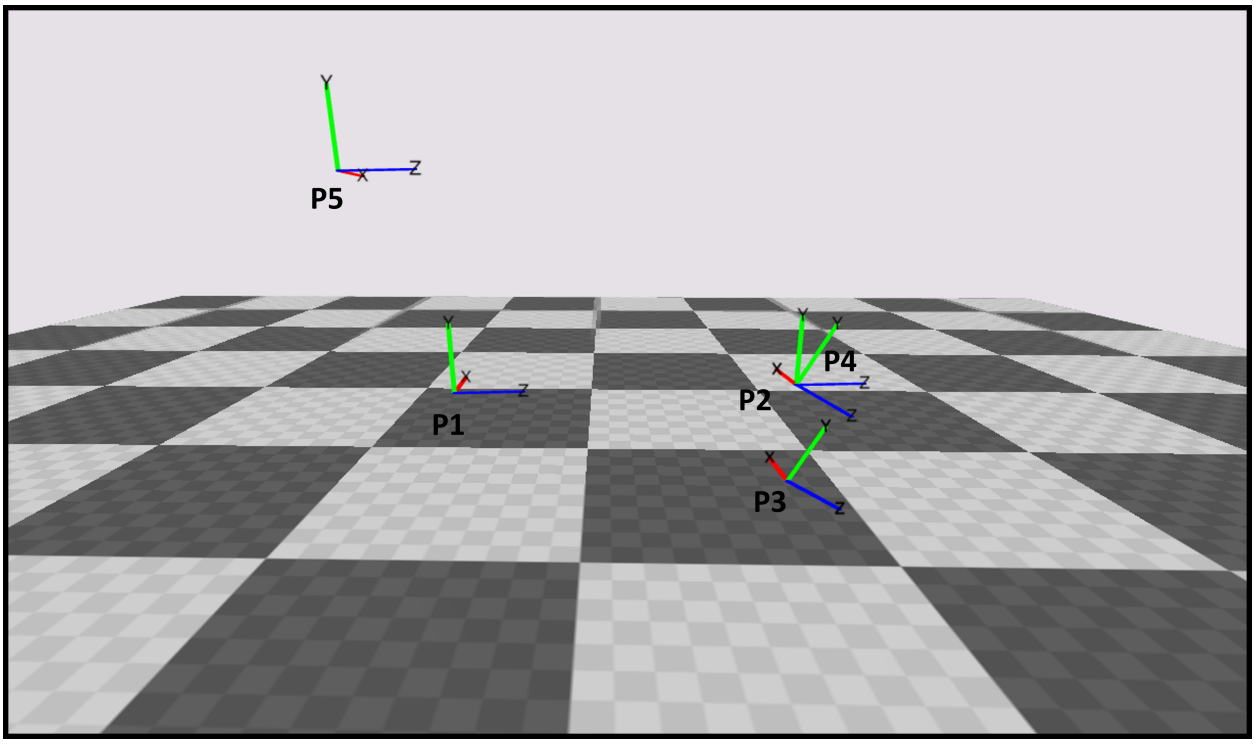


Figure 5.7: This is the point set that will be used in the reachability experiment. Points 1-4 are within the robot's reach and point 5 is not.

the end effector usage, the EPM has a variable cycle time depending on the charging and communication. In these experiments, an estimate will be used from the data.

5.5.4 Experiment Details

The following experiments will focus on the four different ability analysis points and give an example of how they can be implemented.

Reachability Experiment: To evaluate the reachability analysis, five different points were chosen. Pictured in the workspace graphic in Figure 5.7, four of the five points, $\{P_1, P_2, P_3, P_4\}$ are ones that should be reachable within the joint limit bounds. The fifth point, P_5, will require the arm to go out of its bounds to reach it.

Process Time Experiment: To evaluate the processing time modeling, the four valid points, defined above, were used in an operation set to deliver a component to a designated area. The process, pictured in Figure 5.8 Panel A consisted of 5 steps defined as follows:

1. Move the robot to a known position for consistent analysis (the starting point).
2. Move the component to a goal position over the target area.
3. Move the component to the a goal position just over the target area, the drop off point.
4. Trigger the EMP end effector to release the component into the target area.
5. Move the arm out of the way into a neutral position.

These operations were completed for a large and small component, each fitted with an EPM connection point. Figure 5.8 Panels B - F provide a picture of what each step in the process looked like carrying the larger component.

Load Bearing Experiment: To evaluate the load bearing, the joint torques applied to the arm portion of the MARC (joints 4-7) were evaluated carrying the heaviest component in the experiment set. To include weights of the arm links and the end effector, the masses were estimated to values given in Table 5.2. The masses were applied according to Figure 5.9.

Table 5.2: Mass estimations for MARC arm.

Link	Mass (g)
2	1505
3	1505
4	775

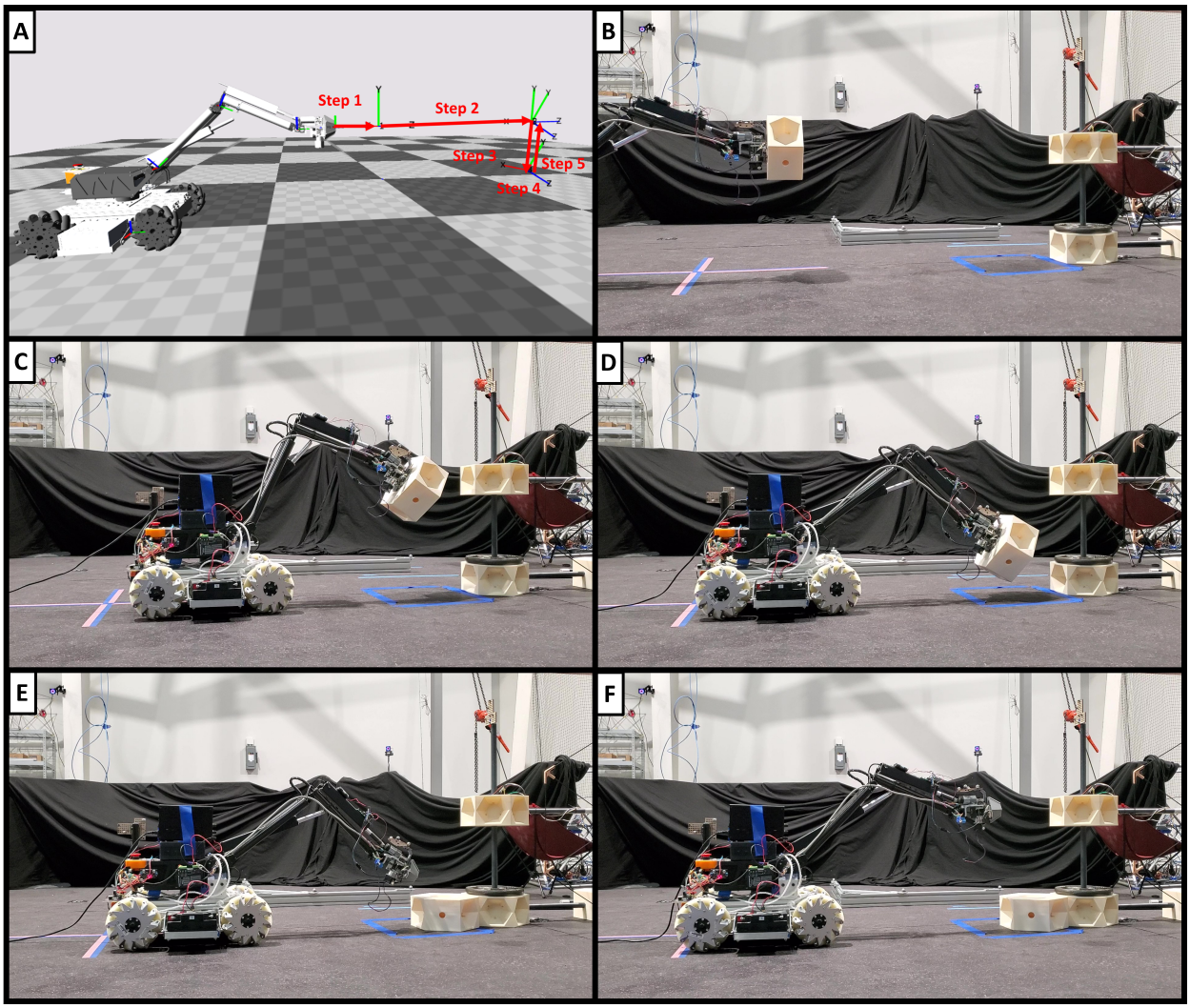


Figure 5.8: Panel A: Steps required to complete task. Panel B: At the completion point of step 1 (the beginning point of the experiment). Panel C: At the completion point of step 2 (component above deliver point). Panel D: At completion point of step 3 (component moved down to release point). Panel E: At the completion point of step 4 (the EPM was triggered to release the component). Panel F: At the completion point of step 5 (the arm moved to the finishing point of the task).

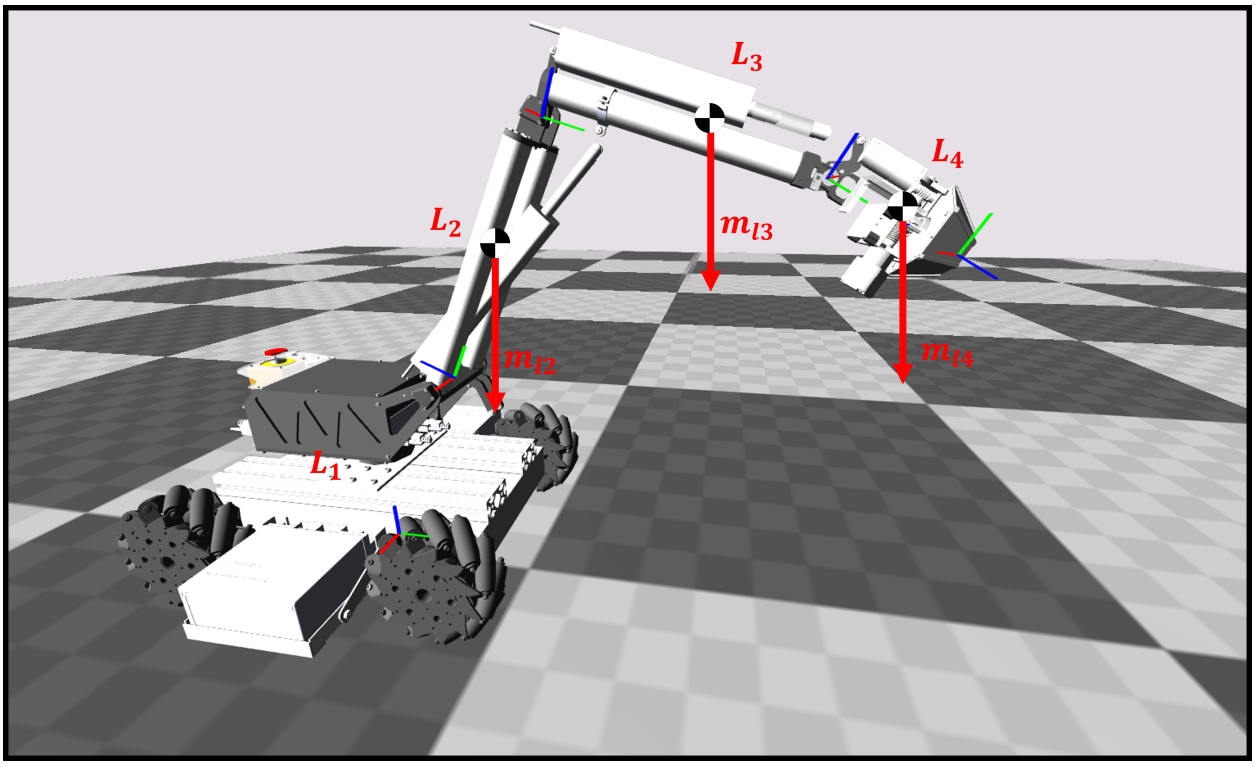


Figure 5.9: Mass locations for static load analysis for MARC arm.

Probability of Failure Experiment: In this section, an estimate was found for the probability that the robot end effector would not be within the desired tolerance to deliver the component. The delivery area for this experiment set is a $48cm \times 54cm$ region centered under the position of goal point 3. Two different tolerance volumes will be considered for the sake of this analysis. Both will be defined as rectangular cuboids centered about P_3. The first, V_{t1}, has the width, depth, and height $[10cm, 14cm, 2cm]$. The second, V_{t2}, defined by the dimensions $[2.5cm, 3.5cm, 0.5cm]$. These two volumes can be seen in the simulation workspace in Figure 5.10 in Panels A and B where the larger and smaller volumes can be seen in green and purple respectively.

The variance for each joint used in this work were found looking at the minimum response resolution of each joint's actuator and are defined as:

$$\Sigma_\theta = \mathbf{diag}([0.005m, 0.006m, 1.15^o, 0.8^o, 0.8^o, 0.8^o, 0.64^o]^2)$$

To estimate the probability described by Equations 5.32 and 5.33, a Monte Carlo approach was used. First, $100,000$ positions were sampled from the multivariate distribution defined in Equation 5.32:

$$X \sim \mathcal{N}(\mu, \Sigma_{ee}) \tag{5.34}$$

Each sample in X was then evaluated to determine if it was inside the tolerance volume. The number of points inside divided by the total number sampled to approximate the probability defined in Equation 5.33:

$$P_{success} \approx \frac{1}{N}\sum_{i=1}^{N} f(X_i) \quad \begin{cases} f(X_i) = 1, & \text{if } X_i \in V_{tolerance} \\ f(X_i) = 0, & \text{otherwise} \end{cases} \tag{5.35}$$

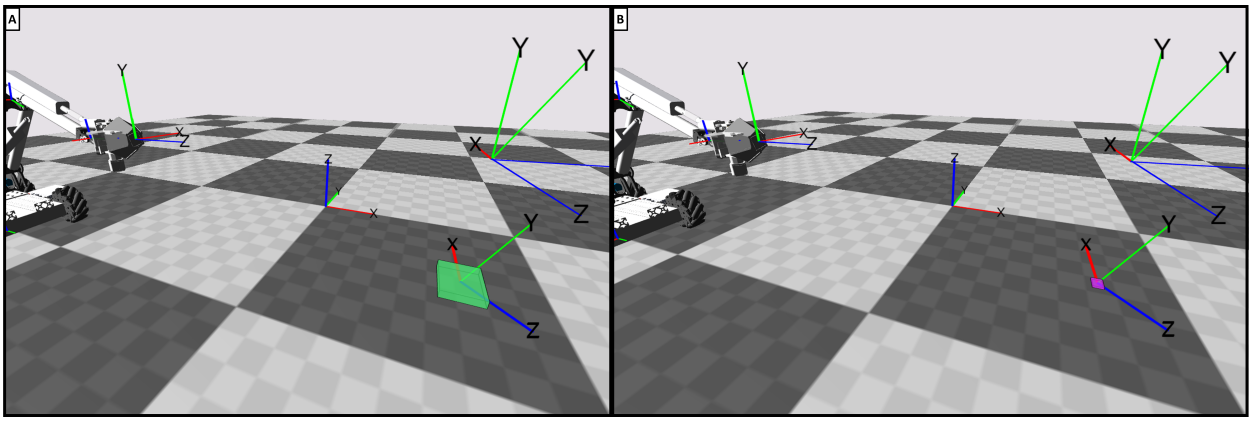


Figure 5.10: Panel A: Larger tolerance box for positional criteria of success. Panel B: Smaller tolerance box for positional criteria of success.

where N is the number of samples and $f(\cdot)$ is a function that evaluates if a point is within the tolerance volume. Since both tolerance volumes in this experiment are rectangular cuboids, this function found the vector difference between the corner of the volume and the point X_i, evaluating if any of the vector lengths were outside the volumes dimensions.

5.5.5 Results

The following sections contain the results for the experiments focused on the different ability analysis metrics.

Reachability Experiment: The algorithm correctly identified that, to reach the fifth point, the arm would have to violate the joint bounds. To visualize this, P_5 was specifically chosen such that the arm "could" reach the point in simulation but only in a way that violated the bound limits. Algorithm 5 did return a configuration (shown in Figure 5.11) but it also noted that the solution was non-viable with a return of *false* for P_5.

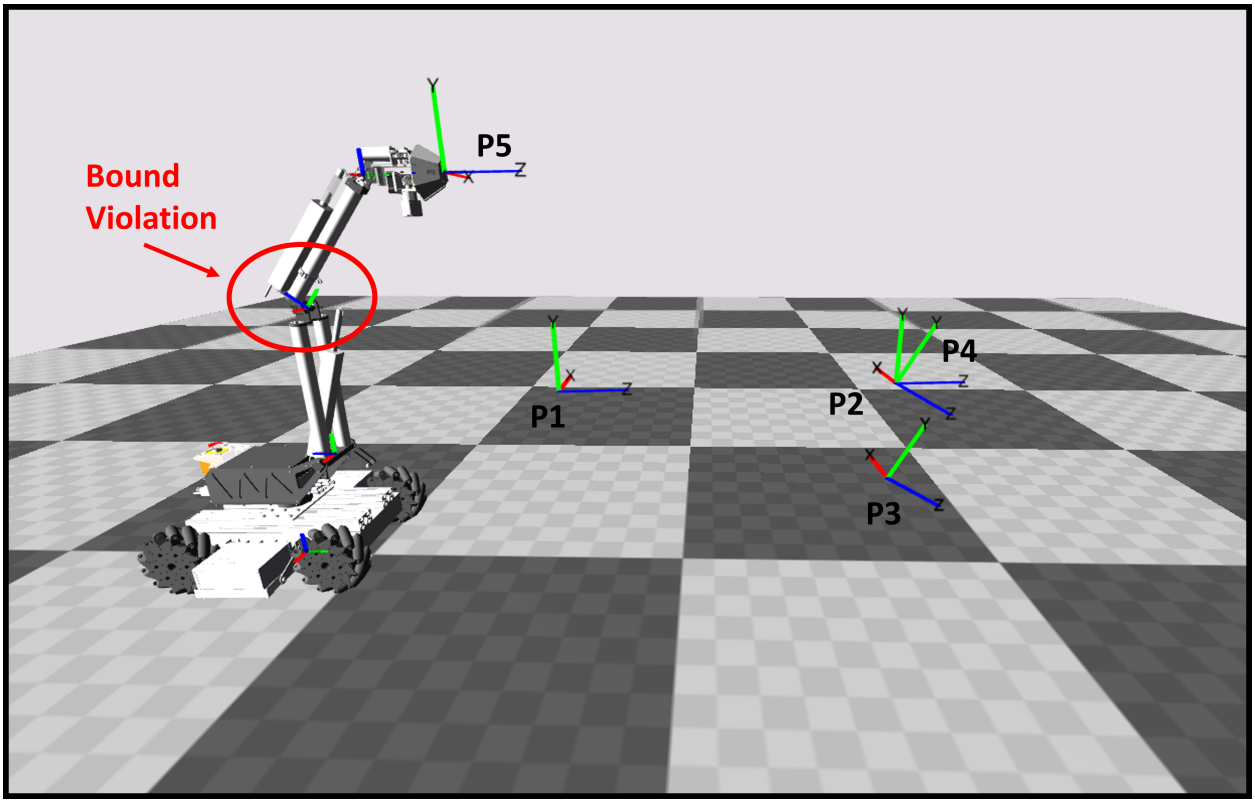


Figure 5.11: In the reachability experiment, P_5 could not be reached without violating the elbow joint bound.

Processing Time Experiment: To analyze the system processing time experiment the results are broken into two different portions, as discussed above, when it comes to estimating the time spent following a path $t^i_{\Delta p}$. The first portion is when the arm follows the max velocity cap for the joints as the control law. The second portion is defined by the converging control model also described in Section 5.5.3. The latter case is going to result in a much larger error than the first due to the nature of the unpredictability. As a result of dynamic scaling, it is possible the robot will need to repeat the convergence step more than once. The results from both portions will be mentioned below with the primary focus being on the pre-convergence state. In addition to the motion time estimate, the time it took to process at a point, t^i_p, was the time it took for the EPM end effector to trigger OFF. This also had a large variability and depended on the electronics and internal communication of that specific subsystem. Since there was not a good theoretical model present for both the convergence and the EPM times, distributions were generated from the actual processing times and the mean values were used to predict the time spent. These distributions can be seen in Figure 5.12.

Before the different elements are analyzed separately, the full set of data can be seen in Figure 5.13. The log-log scatter plot on the left shows how well the prediction aligns with the true values. If the prediction is perfect, each point will fall exactly on the diagonal red line. To the right / lower side of the line indicates that the prediction was shorter than the actual time and to the left / upper side of the line indicates that the prediction was longer than the actual time. The right graph shows the percent error in log form. It is clear that there are different spreads within the data. Some data points are tightly clumped and others have a very large percent error.

As discussed earlier, there is going to be quite a bit of error present in the convergence time. The difference in the predictions are most apparent in step 2, shown in Figure 5.14. Panel A shows that most of the error in the prediction is within 10% whereas the percent error

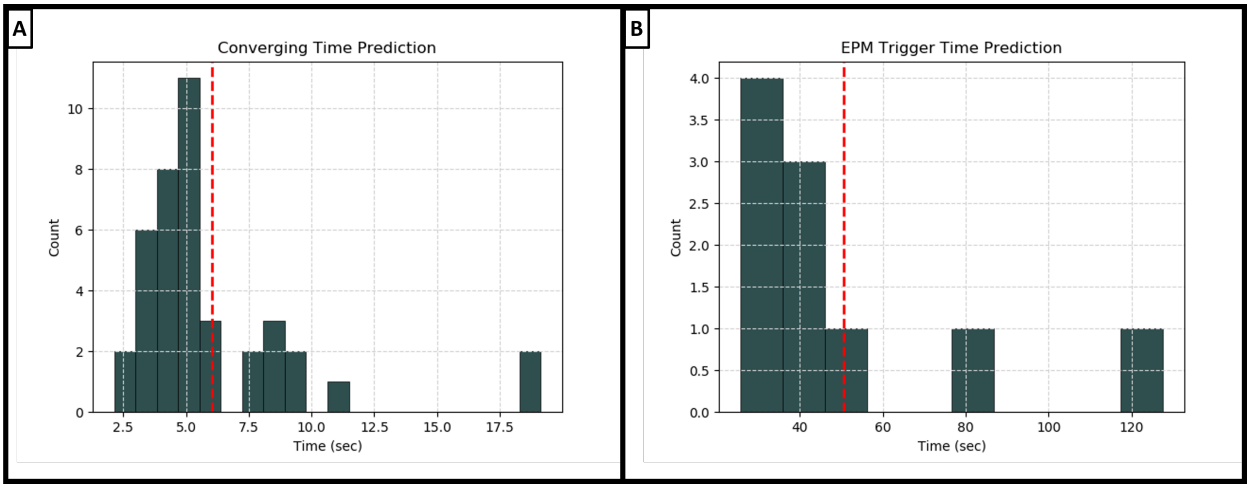


Figure 5.12: Panel A: The distribution of converging time used for prediction. Panel B: The distribution of EPM times used for prediction. The mean for the distributions are represented by the red dashed lines.

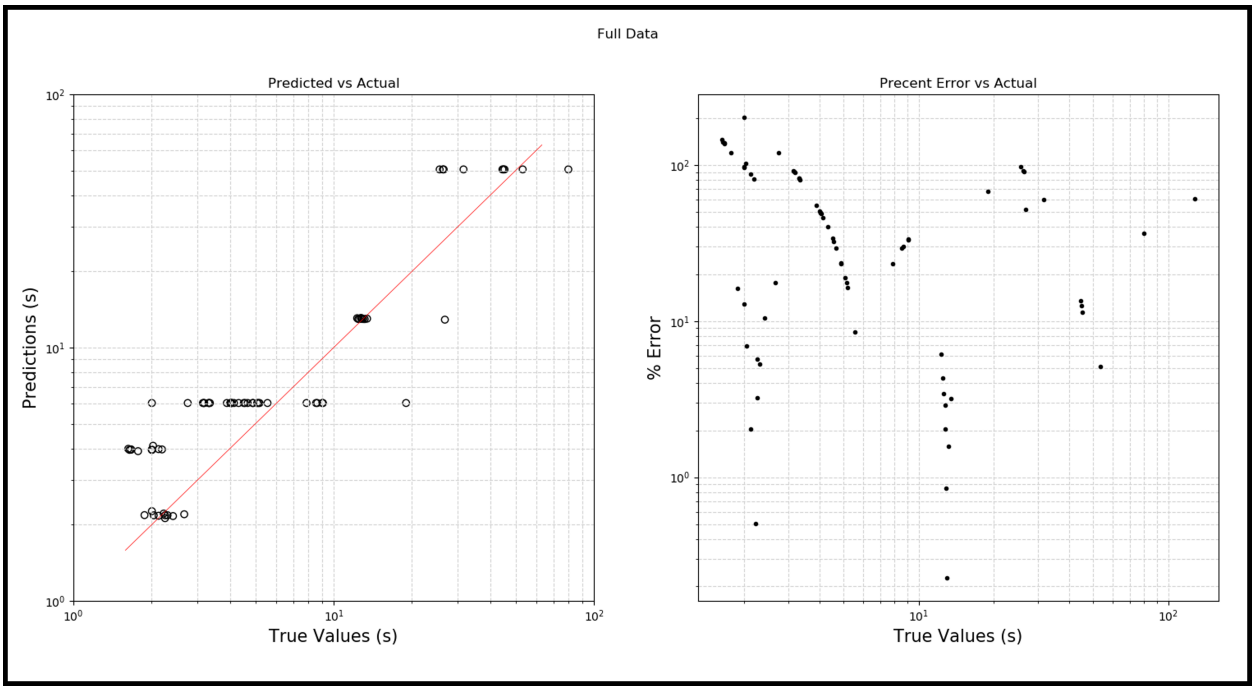


Figure 5.13: The left scatter plot shows the log-log plot of the prediction vs. the true values. The right scatter plot shows the log percent error for each of the true values.

for the portion using the data mean as prediction (Panel B) is between the 20% and 140% range. To walk through each of the different steps, the pre-convergence data is going to be used since it is directly related to the kinematic model. The graphs for the convergence time for steps 2, 3, and 5 (step 4 is the EPM and therefore not applicable) are available in Figure E.1 in Appendix E.

While the processing time for step 2 was very tightly grouped, the prediction for step 3 was consistently low (Figure 5.15 Panel B). This lead to a very high percent error. Part of the reason for the high percent error is due to the small time window that it took step 3 to occur. The actual motion only took 2 seconds. Since the prediction was right around 4 seconds the percent error is around 100%.

Panel A in Figure 5.16 shows the data for the EPM triggering. As mentioned earlier, the EPM has a 100 second range in the processing times. As such, the percent error is going to be higher. Step 5, given in Panel B of the same figure, once again shows a tighter grouping between the prediction and the actual values, hovering again around 10%.

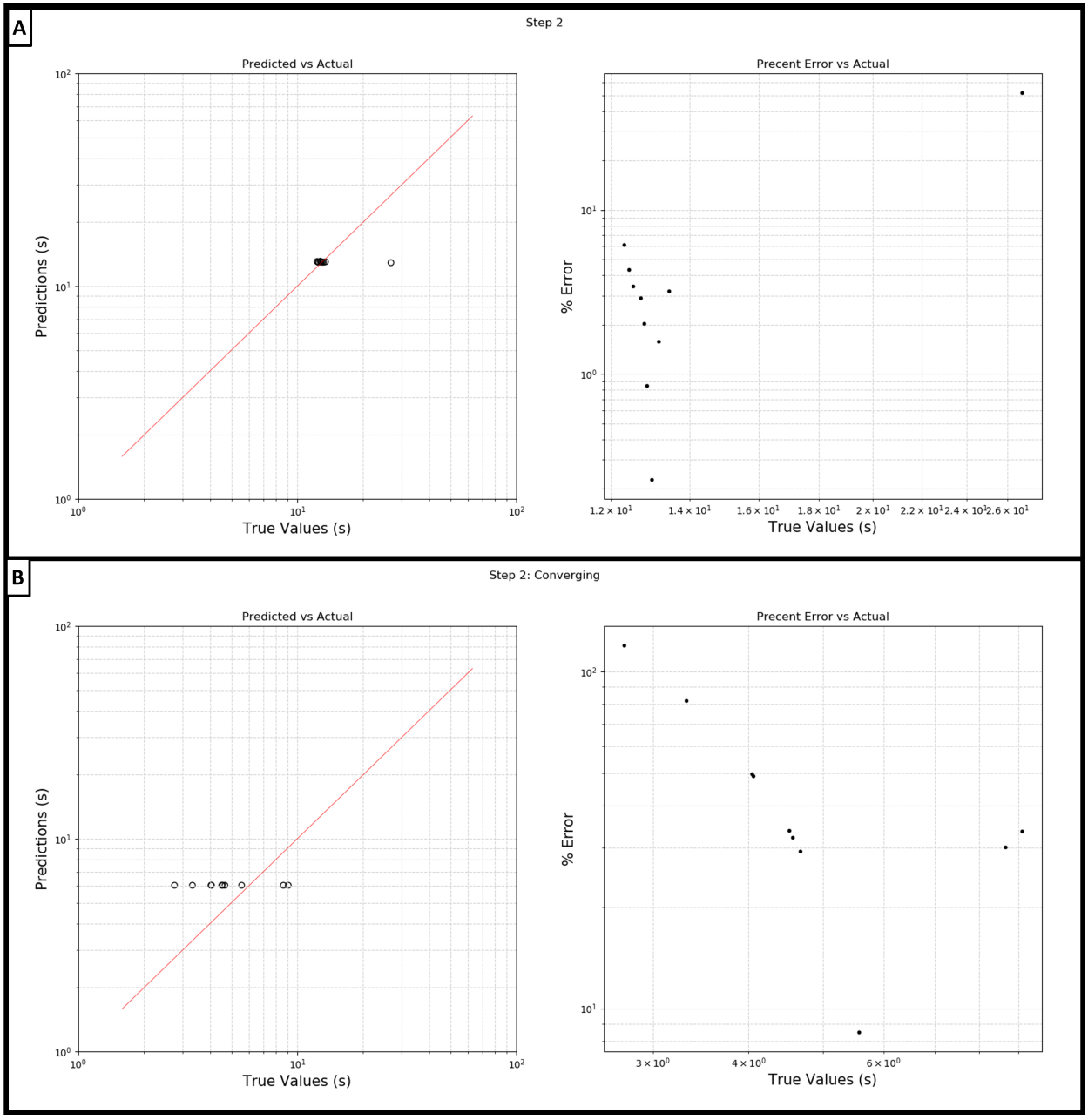


Figure 5.14: Panel A: The log-log prediction vs. true value plot and the percent error plot covering the range of motion in step 2 before the robot reaches the converging point. Panel B: The log-log prediction vs. true value plot and the percent error plot containing the data from the converging portion of step 2.

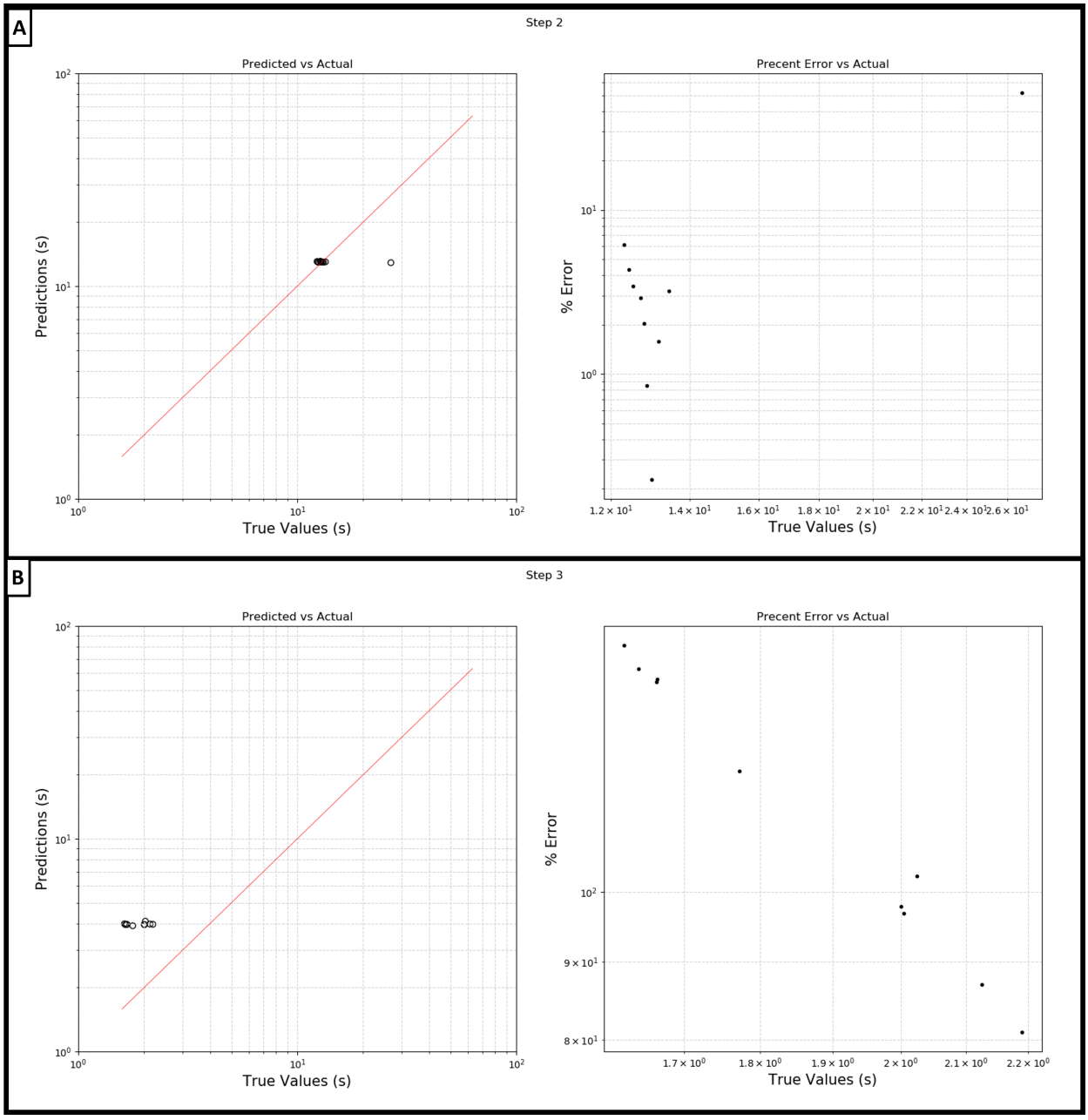


Figure 5.15: Panel A: Is the log-log prediction vs. true value and percent error plots for the step 2 data before it reaches the point where the converging controller engages. Panel B: Is the log-log prediction vs. true value and percent error plots for the step 3 data before the arm reaches the point where the converging controller engages.

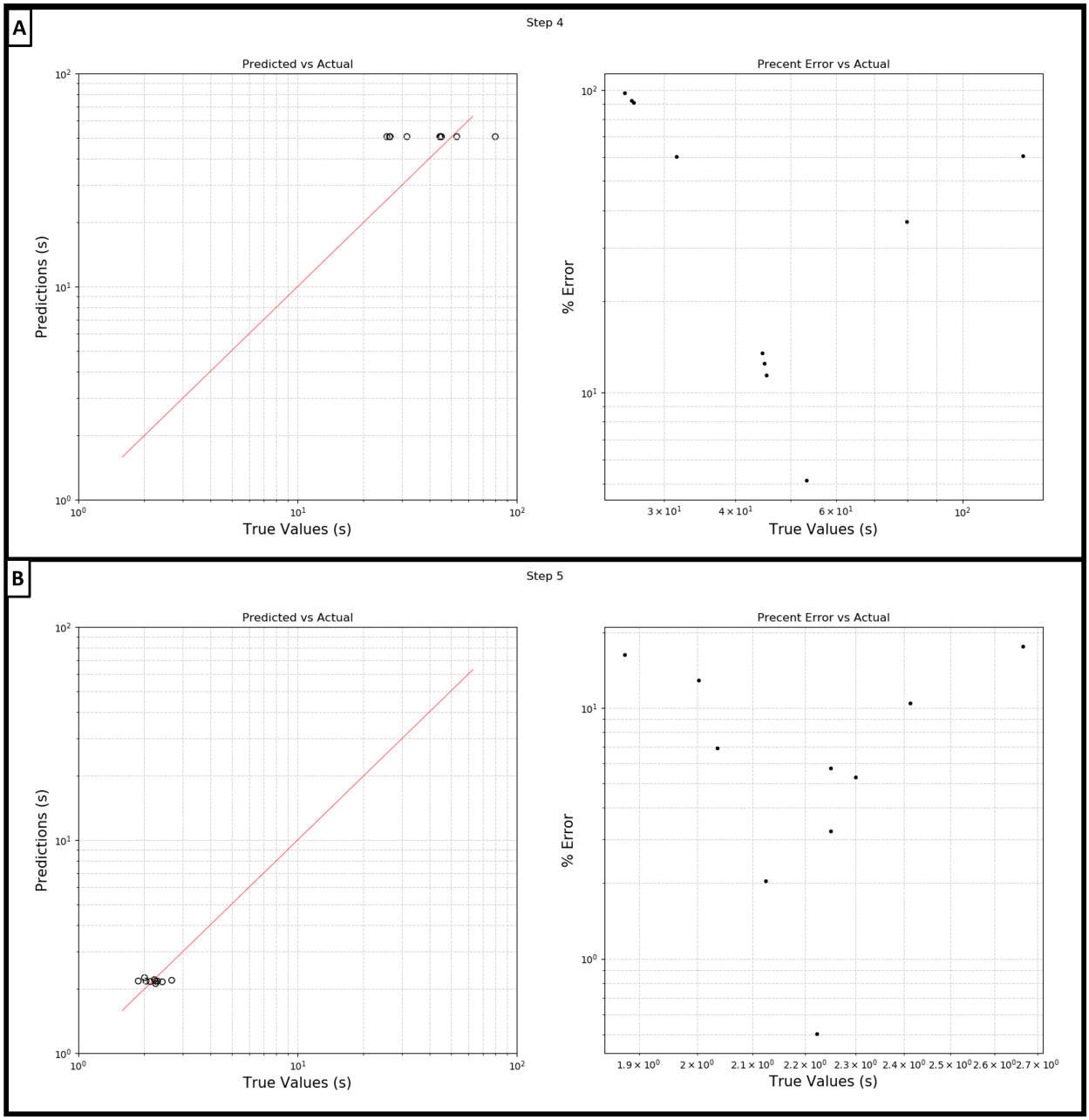


Figure 5.16: Panel A: The log-log prediction vs. true value and the percent error plots for step 4. Panel B: the log-log prediction vs true value and the percent error plots for step 5.

Load Bearing Experiment: The joint torques were calculated for each of the load bearing points in the experiment series. Each of the torque sets were compared against this project's torque limits for joints 4-7: $\tau_{max} = [\ |25|Nm,\ |25|Nm,\ |25|Nm,\ |3|Nm]$. It was found that each of the points did not violate the limits. The results are summarized in Table 5.3.

Table 5.3: Results for static torque analysis.

Point	Joint 4 (Nm)	Joint 5 (Nm)	Joint 6 (Nm)	Joint 7 (Nm)	Payload	Status
1	5.57	-18.39	16.83	0.19	1050	true
2	18.48	-17.94	18.12	0.19	1050	true
3	10.93	-21.17	18.13	0.12	1050	true
4	10.93	-21.17	18.13	0.12	1050	true
5	3.34	-12.49	7.13	0.0	0.0	true

In the results, the directions of the torques are defined by the directions of the screws. The difference in the signs is a result of the screws pointing different directions as shown in Section 5.5.1. It should also be noted that there is a slight torque on the joint 7 motor when there is a payload. This is a result of the a slight offset in the frames for the end frame and the final joint due to the measured numbers and variability in the hardware (as discussed earlier, this is a hardware system in development). For each of the projected goal poses, the torque was found to be within the project tolerance.

Probability of Failure Experiment: Using the covariance and Monte Carlo estimation described in Sections 5.4.4 and 5.5.4 for the two different size tolerance volumes, it was found that the end effector would be within the larger volume with 0.71 probability - a 29% chance that a minor failure, the end effector not being within the tolerance volume, would occur. For the smaller volume, the estimated probability of success was 0.37, a 63% chance the robot would be out of tolerance. These results are represented in Figure 5.17 were the

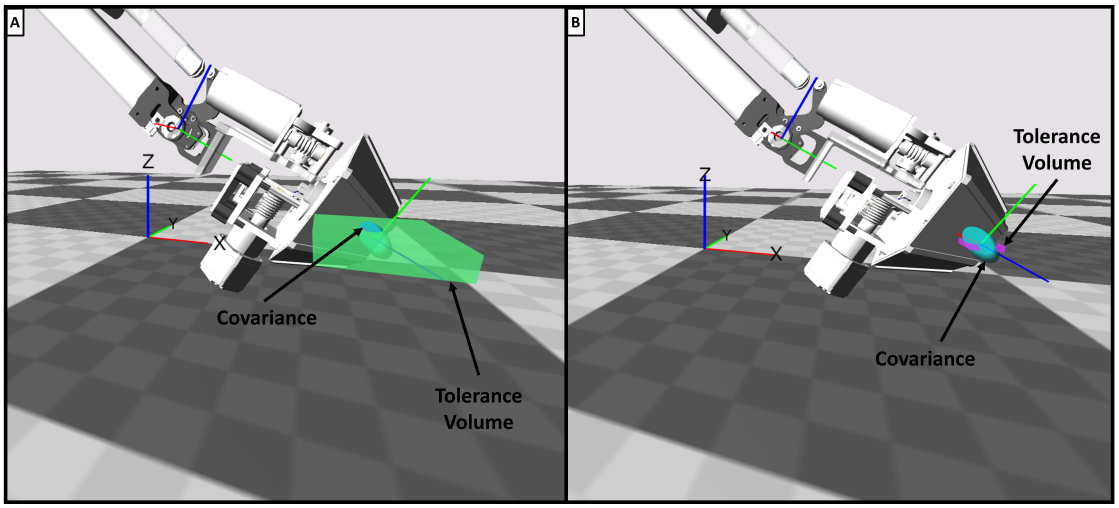


Figure 5.17: Panel A: End effector covariance and larger tolerance volume. Panel B: End effector covariance and smaller tolerance volume.

turquoise ellipse represents Σ_{ee} and the green and purple volumes are the larger and smaller tolerance volumes in Panel A and Panel B (respectively).

5.5.6 Discussion

In the processing time prediction vs. actual times, the difference in the prediction between step 3 and step 5 is interesting since the robot arm is traversing the same distance. The difference in the prediction time (4 seconds for step 3 and 2 seconds for step 5) comes form the difference in the twist. The wrist needs to be kept at a specific orientation with respect to the spatial frame in step 3. As such, the twist vector is different since it includes wrist change as well. In step 5, the wrist is returning to a parallel orientation for the end effector and space frame axes. For this motion, the wrist does not need to move since the lifting of the arm through the shoulder actuation will move the wrist into the correct orientation.

While the better predictions were within 10% of the error, the portions that did not have a good model were farther off by a much larger percentage. If a modelable controller was

in place for the converging portion, this error would be reduced. However, in a real world scenario, it is not unrealistic to have portions, like the EPM and the convergence controller portion, where the time can not be accurately predicted. Such unknown portions highlight the need for a system that is capable of adapting through replanning and reallocating tasks to keep an assembly schedule as optimal as possible when things stray from predicted values.

The static analysis of the goal points indicated that the estimated goal pose for each of the points in the project set were within the project bounds for the largest payload. As discussed, future work can focus on developing a dynamics model for the robot as the hardware becomes more stable and the control formulation is completed. These advancements will allow for a more accurate prediction as it will be able to include the inertial components as the robot passes through the different points in the path.

The probability analysis estimated a failure probability based on the uncertainty in the input resolution and found a 29% chance the end effector would be outside the larger tolerance volume and a 63% chance the robot would be outside the smaller tolerance volume. Future work can look at including additional factors that contribute to the chance of failure.

5.6 Summary of Contribution

The ability analysis presented in this chapter addresses the third research question inquiring about a method for estimating robot ability when the hardware can not be experimentally tested. In this chapter, four different ability criteria are discussed as a framework that specific problem instances and kinematic models can use to estimate if and how a robot can complete a task.

For the kinematic portions, a screw theory kinematic formulation is discussed to highlight

the use of the metrics and then utilized them in an example. Additionally, this chapter discussed how more complex modeling frameworks that were not present in the experiments (such as a dynamics model or complex controllers) would fit in to the proposed metrics. A framework for how to think about the probability was also proposed, with a discussion on some considerations for classifying the severity of failure.

Chapter 6

Assignment Generation Selection Metrics

Different types of schedule generation formulations preform differently when it comes to traits such as schedule optimality, solving time, computational power requirement, and proof of optimality to name a few. This range of characteristics leads to questions of how to choose a scheduling method and what are the metrics that should be used in the processes of choosing one scheduling method over another? This chapter will discuss and develop a metric set that can be used to help choose a scheduling method for the in-space assembly problem.

6.1 Research Gap

In the literature, it is common to compare algorithms against one another for the purpose of determining if one is superior over another. Recent literature surveys such as [20, 63] list existing criteria for comparison that include efficiency (the measure of fundamental evaluations within an algorithm or the running time). This can be estimated a number of ways that include using the average run times [39], the median run times with quartile information [22], or a completion time ratio [15]. Reliability is another comparison method, focusing how well the algorithms preform over a wide range of problems through the use of metrics such as success rate, evaluating the average objective function output [129], or

the distance of solutions from a bound. The quality of the output is another consideration, measuring how good the generated solution is based on known solutions or calculated bounds [20].

6.2 Metrics

The metrics in the literature compare one optimization method against another when evaluating their capability. However, the literature does not directly provide insight into which scheduling metrics help determine the best scheduling method for different in-space assembly scenarios.

In the autonomous in-space assembly problem where an autonomous system is operating without human input, there are a range of different scenarios that arise. The first is the primary development of a schedule with known features. This scenario covers the conditions of determining how a robotic system should attempt to assemble a structure at the beginning of the process, the blueprint upon deployment. In this case, it is reasonable that a lot of computational power could be put towards the scheduling problem to find a solution that was optimal, potentially optimizing across multiple objectives.

Another such scenario would be a case where an assembly was required that was not specifically planned out but the timing is less critical. An example of this would be if a robot goes down during a non-time-critical assembly. The optimal schedule would potentially change due to the robotic workforce changing but there is still the potential for the assembly to be paused, allowing for a more computationally expensive optimization method to take place.

In contrast, another plausible scenario for a fully autonomous robotic assembly workforce to encounter is to have something go wrong in an assembly that must be repaired quickly.

In such a scenario there may not be the time to allow for a thorough, computationally expensive optimization process to take place. In this situation, a fast and efficient schedule is necessary but a long optimization time can not be afforded or may result in a longer delay in the finishing of the assembly when computation time is taken into account.

Another factor that can impact the selection of a scheduling method is the type of information available and pertinent to the assembly. For example, if the tasks and robots are such that some tasks have a high chance of failure if attempted by certain robots in the workforce, a scheduling system that can handle the stochastic information would be desired. Alternatively, if the stochastic information is not known for an assembly, less complex scheduling methods may be sufficient to find an optimized solution.

This work proposes three primary metrics that utilize the problem formulation developed in Chapter 2 and the existing literature to provide a framework for making a decision on what scheduling method to choose. It should be noted that these are not metrics meant to only use information calculated during deployment by the autonomous system. They are meant to be tools to evaluate different methods to be assigned to given scenarios that may be encountered after deployment.

6.2.1 Explanatory Power

The first metric is *explanatory power*. This metric is meant to evaluate how much of the information present in the assembly problem can be incorporated into the schedule generation process. In the SAPD, the different features will be classified into six categories:

1. Task Proximity

2. Task Precedence

3. Task Continuity
4. Robot Ability
5. Stochastic Processing Time
6. Stochastic Failure

These categories break down the constraint considerations into three different groups and the completion ability into three different groups. *Task Proximity* refers to the scheduling method's ability to incorporate the distance between tasks into the scheduling process. In a similar way, *Task Precedence* and *Task Continuity* are describing the scheduling method's ability to take into account these two sets of constraints respectively. *Robot Ability* is the metric that seeks to evaluate if a generation method can include a measure of how well the robot can complete a job. This is not to be confused with the *Stochastic Processing Time*. The stochastic nature is a separate metric since there is a difference between being able to take the robot ability into account and being able to account for the stochastic nature. The final category is *Stochastic Failure* which, as the name implies, is the ability of the system to take into account the chance of failure.

These six criteria can be thought of as a vector of binary variables representing if each one of the six are accounted for in the model (where 1 means it is taken into account and 0 means it is not): $Ep = [ep_1, \ldots, ep_n], ep_i \in \{0, 1\}$ where $n = 6$ in the case where all elements in the SAPD are being considered.

Due to the wide variety of assembly problems, it is possible that all of these metrics are not weighted equally. For example, if there is a high chance of failure it might be more important to be able to model the stochastic elements rather than the time taken up by the distance robots have to travel. In this case, it would not be desirable for a generation method that

did take into account distance but was missing stochastic failure to have the same score as a method that included the stochastic failure but left out the distance consideration. To provide a way to tune the importance of each of the criteria, a weight vector can be applied where each weight corresponds to how much importance a given parameter has: $Wep = [w_1, \ldots, w_n]$. If the weighted vector is divided by its magnitude, the full form of the explanatory power metric becomes $M_{Ep} \in [0, 1]$:

$$M_{Ep} = \frac{Ep \cdot Wep}{|Wep|} \tag{6.1}$$

6.2.2 Utility

The second metric when considering a scheduling method is *utility*. In contrast to the previous metric which looked at how well a scheduling method covered the characteristics of a problem, this metric is a consideration that reflects how well a scheduling method can be used in a given scenario. This metric can be modified to account for the computational power and how long the system can wait for a schedule to be generated.

How this metric is calculated can vary depending on the type of schedules or what criteria are used. The general form based on processing time limitations takes the form:

$$M_u = \begin{cases} 1, & f(method, problem) \leq T_{max} \\ 0, & \text{otherwise} \end{cases} \tag{6.2}$$

where T_{max} is the max allowable time for for generating a schedule and $f(\cdot)$ is a function that will map the scheduling method and assembly problem to an approximate solution time. One way to utilize $f(\cdot)$ is to view it as a function that returns the amount of time required to produce the first viable solution. As long as a viable solution can be produced in a time that

is less than T_{max}, it would meet the metric requirement of being able to produce a solution in the necessary time. Another way to interpret this is as an estimation of the complication time based on previous runs. For example, experiments can be done similar to those in Chapter 5 where problems with similar constraint types and job sizes are run on equivalent hardware. This information can then be used to return an estimate of how long it will take to generate a valid solution. Other metrics available in the literature include estimating this using average and cumulative run times [39], or by comparing the median and quartile solver times [22]. If there is a need to compare across different types of computational hardware, formulations can be developed try to estimate the time based on compute cycles [63]. What specific metric is used here is a design consideration based on the information available upon deployment. As long as the selected method produces a measurement that will inform if a solution can be found by the utility requirement, it can be used to satisfy this utility metric.

6.2.3 Optimality

The third metric is the *optimality* evaluation of the scheduling method. This metric is meant to provide a comparison point to evaluate the difference in how good (from an objective function perspective) the solutions are from different scheduling methods. The specific form that this metric can take is another design parameter that can be chosen based on the information known. This can be done using a fixed-cost method measuring against an optimal solution, if it is known, or the best known solution as an estimate [20]. More complex methods can also be chosen that attempt to statistically estimate what the optimal would be as the standard to compare against [44].

For the purpose of this work, this metric is going to be based on how close the objective

value is to the best known solution for the problem:

$$M_{op} = 1 - \frac{||\bar{x} - x^*||}{\bar{x}} \tag{6.3}$$

where $\bar{x}$ is the average objective value and x^* is the objective value for the best known solution for the assembly problem. Since the goal is to minimize the makespan, the more optimal solution will be smaller. As such, the difference is normalized by the larger value. Subtracting this value from 1 gives some score up to 1, where 1 means it found the optimal. This difference be implemented two different ways. The first is that the average value, $\bar{x}$, is used for each solution method after a fixed time span T. If the time is not a concern then the average can be found based on the solutions resulting from the converged state of the schedule generation algorithms. This solution can be defined in the general form as:

$$M_{op} = \begin{cases} 1 - \frac{||\bar{x}_T - x^*||}{\bar{x}}, & \text{Average calculated from solutions after } T \text{ solving time} \\ 1 - \frac{||\bar{x}_c - x^*||}{\bar{x}}, & \text{Average calculated from solutions after optimization converges} \end{cases} \tag{6.4}$$

6.2.4 Metric Application

The weight given to decide on a final scheduling method is a design decision made by those deploying the system. This can be something that is determined ahead of time either by a hard defined list of what types of assemblies and scenarios will use certain schedulers or some algorithm that uses the metrics to evaluate scenarios as they arise and makes the necessary decisions. The following section will give an example of these metrics being used on the mixed integer programming method developed in Chapter 3 and the genetic algorithm

method developed in Chapter 4 for the simplest cases (single arch and 5 x 2 truss wall). The scheduling time limitation for this example is: $T_{max} = 24hr$.

6.3 Metric Implementation Example

To demonstrate the metrics, the arch assembly will be uniformly weighted since there is no stochastic information and the truss block problem will have extra weight placed on the stochastic information. This leads to weight values shown in Table 6.1.

As noted in this table, the arch assembly does not have any stochastic information present in it or the full set of constraints. As such, both the MIP and the GA are able to articulate all of the elements present. However, the truss block wall assembly does have stochastic processing time and all of the constraints. The GA formulation is able to take into account the stochastic information since it can directly sample the ability distributions each run. In contrast, the MIP formulation must rely on the mean of the distributions making it less capable of taking into account the stochastic information. This is reflected in Table 6.2.

Combing the weight and the explanatory power vector, the GA has a M_{E_p} score of 1 for both projects and the MIP has a M_{E_p} score of 1 for the arch project and 0.77 for the truss project. Since the single arch and small wall project are being used in this example and both were able to find a solution in the allowed time, they both have a value of 1 for Mu. For the arch problem, the MIP found the optimal giving it an M_{op} score of 1. The GA also found

Project	Weights					
-	w_1	w_2	w_3	w_4	w_5	w_6
Arch	1	1	0	1	0	0
Truss	1	1	1	1	1.2	0

Table 6.1: Weights indicating the importance of the different features for the arch and truss projects.

Table 6.2: Weights indicating the importance of the different features for the arch and truss projects.

Method	Project	Explanatory Power					
-	-	ep_1	ep_2	ep_3	ep_4	ep_5	ep_6
GA	Arch	1	1	0	1	0	0
MIP	Arch	1	1	0	1	0	0
GA	Truss	1	1	1	1	1	0
MIP	Truss	1	1	1	1	0	0

Table 6.3: The metric values for for the single arch and 5 x 2 truss wall projects.

Project	Method	M_{E_p}	M_u	M_{op}
Arch	GA	1	1	1
Arch	MIP	1	1	1
Truss	GA	1	1	0.85
Truss	MIP	0.77	1	1

the optimal giving it the same score. For the small wall problem, the MIP was not able to prove convergence. However, in this case, it did provide the best known solution (between the two solvers) so it will also have a score of 1. The GA did not find as good of a solution. Using the numbers from Table 4.3, it can be seen that the GA has a M_{op} score of 0.85.

The three metric scores are summed below in Table 6.3. For the arch assembly, both the MIP and GA are equivalent but for the truss wall, there is a difference. The GA is not as good at returning an optimal solution and the MIP is not as good at covering the full problem scope. If the truss wall is being assembled when the resources are low or there are many other projects that the robots need to also complete, it will be more advantageous to use the MIP scheduler (if the stochastic information will not greatly impact the time). However, if the stochastic information contains important implications such as the chance of great delays once the project is started, the GA will be a better choice.

It can be seen that, if there were different considerations such as a much shorter solving time allowed, the metrics would be different. For example, if $T_{max} = 20min$ the MIP might not

score well in the utility metric if it could not find a viable solution in that time frame. If it did find a solution but it was not very good compared to the solution it could find running for $12hr$, then it would score less in the optimality metric.

6.4 Summary of Contribution

This chapter addresses the fourth research question inquiring into a method to compare different optimization methods for the in-space assembly problem. To answer this question, this work develops the explanatory power metric (M_{Ep}) to quantify how much of the assembly problem is being taken into account by a given scheduling method. In addition to this novel metric, it also frames two other metric types seen in the literature. The first evaluates how usable a scheduling method is (M_u), which relates to the solving time in this work, and a measure of how good the average solution generated by the method is (M_{op}) based on how close to the best known solution for the closest assembly problem. Using these metrics, a scheduling method or set of scheduling methods can be chosen based on quantifiable values for specific scenarios.

Chapter 7

Conclusion and Future Work

The work presented in this dissertation addresses the need for a system that enables a robotic workforce to plan and re-plan an assembly through task sequencing and allocation without the need for human input. In this work, a problem formulation was developed to model the different elements that contributed to the assembly problem and provided a framework for articulating the constraints present in the assembly. Additionally, a state representation framework was developed that allowed for the inclusion of stochastic information and was capable of expanding to include the states introduced to correct the errors that are inevitable when autonomous operations occur in real world environments. Future work can expand this formulation model to include additional modeling for the environmental factors that will impact some of the state transitions. Adding this information would allow the schedule generation methods to have a more accurate model of all the contributing factors that will cause the stochastic characteristics present in an assembly.

To address the development of optimized schedules to complete an assembly, this work developed two different schedule generation methods. The first was a mixed integer programming formulation that was capable of generating optimized job sequence and task allocation solutions that were provably optimal or within some quantifiable range of optimality. Future work for this formulation could include expanding it to be multi-objective, allowing the formulation to consider other factors such as energy usage in the optimization process. An additional area of future work for the formulation would be an evaluation of the constraint

formulations to determine if additions to the formulation would aid in the solving time.

The second schedule generation method developed in this work took the form of a genetic algorithm. The novel formulation allowed for existing and newly developed constraint types to be modeled in a way that utilized direct sampling from the ability distributions. This solution method allows for a better representation of the stochastic information present in the problem, specifically if it takes a multimodal form. While it can not provide provable optimal solutions on its own, this method does provide a way to generate an optimized solution very quickly. Future work for this formulation can look at improving the formulation's ability to preserve diversity to aid in the search for the optimal solution. For both schedule generation methods, future work can also include good heuristic methods for providing initial solutions to the problem and thereby speed up the optimization process.

To provide a criteria for how to define robot ability in the context of in-space robotic assembly, this work developed four different metrics and discussed a kinematic formulation to implement them to estimate a given robot's ability. Future work for this would be to provide probability formulations to estimate the contribution of different elements into the chance of failure. This could include error in the position of the end effector due to the error present in the metrology and controller. Additionally, a dynamic model can be added to the kinematic formulation discussed to improve the resolution on the processing time and load bearing estimations. Finally, additional capability instances can be evaluated such as fastening task types.

A metric set was then developed to compare different schedule generation methods in the context of in-space assembly. A formulation was created to quantify how many of the features present in the assembly were utilized by a given scheduling method. It also proposed considerations for solving time and optimality in the decision process. Future work can focus on providing a computation comparison across different hardware configurations, potentially

focusing on computation cycles. Additionally, estimations for optimality potential can also be developed to provide a better measurement of how likely a method is to find the optimal solution.

The methods proposed in this work can be implemented in at least two different planning categories for autonomous in-space assembly. The first is in the planning stage before any robots are deployed. First, a desired assembly is defined in the SAPD. Next, the robot ability information is experimentally determined or analytically approximated. Using this information, an optimized assembly schedule is generated that will serve as the original schedule for assembly. Design decisions are made for the weights used in the schedule comparison metrics and then the system can be deployed with the rest of the autonomous capabilities that will be present for in-space assembly.

The second category of use is after deployment. Once deployed, if something goes wrong with the assembly or one of the robots, the robot ability can be re-estimated (if needed) and an appropriate scheduling method is chosen based on the metric weights set for the scenario at hand (based on the design decisions). A new schedule is generated to repair and / or complete the assembly. This process can be repeated, as needed, until the assembly is complete.

This dissertation demonstrates that there is a way to discover optimal or near optimal solutions to the in-space assembly problem that include considerations related to robot ability and constraints within the assembly structure. By addressing the proposed research questions focusing on problem formulation, schedule generation, robot ability, and schedule generation method selection, this work provides a framework to allow autonomous robots to plan and re-plan without the need for human input.

Bibliography

[1] Altius space machines. URL https://altius-space.com/technologies.html. Accessed: 2022-04-19.

[2] Field and space experimental robotics laboratory. URL https://faser.me.vt.edu/facilities.html. Accessed: 2022-04-18.

[3] In-Space Robotic Assembly of Large Telescopes.

[4] Motion capture systems, February 2019. URL files/101/optitrack.com.html.

[5] Alphonsus Adu-Bredu, Zhen Zeng, Neha Pusalkar, and Odest Chadwicke Jenkins. Elephants don't pack groceries: Robot task planning for low entropy belief states. *IEEE Robotics and Automation Letters*, 7(1):25–32, jan 2022. doi: 10.1109/lra.2021.3116327.

[6] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins SRI, Anthony Barrett, Dave Christianson, et al. Pddl| the planning domain definition language. *Technical Report, Tech. Rep.*, 1998.

[7] Semra Ağralı, Z. Caner Taşkın, and A. Tamer Ünal. Employee scheduling in service industries with flexible employee availability and demand. *Omega*, 66:159–169, jan 2017. doi: 10.1016/j.omega.2016.03.001.

[8] Syed Ali Ajwad, Muhammad Imran Ullah, Raza Ul Islam, and Jamshed Iqbal. Modeling robotic arms–a review and derivation of screw theory based kinematics. In *International Conference on Engineering and Emerging Technologies/Lahore, Pakistan*, page 98, 2014.

[9] B Danette Allen. Osam: Autonomy & dexterous robots. In *Workshop: Logistics and Manufacturing Under Attack*, 2021.

[10] M. R. Amin-Naseri and Ahmad J. Afshari. A hybrid genetic algorithm for integrated process planning and scheduling problem with precedence constraints. *The International Journal of Advanced Manufacturing Technology*, 59(1-4):273–287, jul 2011. doi: 10.1007/s00170-011-3488-y.

[11] T. Arai, H. Izawa, Y. Maeda, H. Kikuchi, H. Ogawa, and M. Sugi. Real-time task decomposition and allocation for a multi-agent robotic assembly cell. In *Proceedings of the IEEE International Symposium onAssembly and Task Planning, 2003.* IEEE, 2013. doi: 10.1109/isatp.2003.1217185.

[12] ChiKit Au, Joshua Barnett, Shen Hin Lim, and Mike Duke. Workspace analysis of cartesian robot system for kiwifruit harvesting. *Industrial Robot: the international journal of robotics research and application*, 47(4):503–510, may 2020. doi: 10.1108/ir-12-2019-0255.

[13] Andrea Zambelli Bais, Sebastian Erhart, Luca Zaccarian, and Sandra Hirche. Dynamic load distribution in cooperative manipulation tasks. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, sep 2015. doi: 10.1109/iros.2015.7353699.

[14] Stephen Balakirsky. Ontology based action planning and verification for agile manufacturing. *Robotics and Computer-Integrated Manufacturing*, 33:21–28, jun 2015. doi: 10.1016/j.rcim.2014.08.011.

[15] Richard S. Barr, Bruce L. Golden, James P. Kelly, Mauricio G. C. Resende, and William R. Stewart. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1(1):9–32, sep 1995. doi: 10.1007/bf02430363.

[16] Javier Barreiro, Matthew Boyce, Minh Do, Jeremy Frank, Michael Iatauro, Tatiana Kichkaylo, Paul Morris, James Ong, Emilio Remolina, Tristan Smith, et al. Europa: A platform for ai planning, scheduling, constraint programming, and optimization. *4th International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS)*, 2012.

[17] Adil Baykasoglu. Linguistic-based meta-heuristic optimization model for flexible job shop scheduling. *International Journal of Production Research*, 40(17):4523–4543, jan 2002. doi: 10.1080/00207540210147043.

[18] Adl Baykasoğlu, Alİ İhsan Sönmez, et al. Using multiple objective tabu search and grammars to model and solve multi-objective flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, 15(6):777–785, 2004.

[19] Behnam Behdani and J. Cole Smith. An integer-programming-based approach to the close-enough traveling salesman problem. *INFORMS Journal on Computing*, 26(3): 415–432, aug 2014. doi: 10.1287/ijoc.2013.0574.

[20] Vahid Beiranvand, Warren Hare, and Yves Lucet. Best practices for comparing optimization algorithms. *Optimization and Engineering*, 18(4):815–848, sep 2017. doi: 10.1007/s11081-017-9366-1.

[21] Wendel K. Belvin, William R. Doggett, Judith J. Watson, John T. Dorsey, Jay E. Warren, Thomas C. Jones, Erik E. Komendera, Troy Mann, and Lynn M. Bowman. In-Space Structural Assembly: Applications and Technology. In *3rd AIAA Spacecraft Structures Conference*. American Institute of Aeronautics and Astronautics, 2016. doi: 10.2514/6.2016-2163. URL https://arc.aiaa.org/doi/abs/10.2514/6.2016-2163.

[22] Ingrid Bongartz, Andrew R. Conn, Nicholas I. M. Gould, Michael A. Saunders, and

Philippe L. Toint. A numerical comparison between the lancelot and minos packages for large-scale constrained optimization. 1997.

[23] Kyle E. C. Booth, Goldie Nejat, and J. Christopher Beck. A Constraint Programming Approach to Multi-Robot Task Allocation and Scheduling in Retirement Homes. In Michel Rueher, editor, *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 539–555, Cham, 2016. Springer International Publishing. ISBN 978-3-319-44953-1. doi: 10.1007/978-3-319-44953-1_34.

[24] Daniella Brovkina and Oliver Riedel. Graph-based data model for assembly-specific capability description for fully automated assembly line design. In *2020 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*. IEEE, oct 2020. doi: 10.1109/ecice50847.2020.9301960.

[25] B. L. Brumitt and A. Stentz. GRAMMPS: a generalized mission planner for multiple mobile robots in unstructured environments. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, volume 2, pages 1564–1571 vol.2, May 1998. doi: 10.1109/ROBOT.1998.677360.

[26] Rafael C. Cardoso, Angelo Ferrando, Daniela Briola, Claudio Menghi, and Tobias Ahlbrecht. Agents and robots for reliable engineered autonomy:a perspective from the organisers of AREA 2020. *Journal of Sensor and Actuator Networks*, 10(2):33, may 2021. doi: 10.3390/jsan10020033.

[27] Yaniel Carreno, Èric Pairet, Yvan Petillot, and Ronald PA Petrick. Task allocation strategy for heterogeneous robot teams in offshore missions. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 222–230, 2020.

[28] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras. Rosplan: Planning in the robot operating system. *Proceedings of the International Conference on Automated Planning and Scheduling*, 25(1):333–341, Apr. 2015. URL https://ojs.aaai.org/index.php/ICAPS/article/view/13699.

[29] Cihan Çetinkaya, Ismail Karaoglan, and Hadi Gökçen. Two-stage vehicle routing problem with arc time windows: A mixed integer programming formulation and a heuristic approach. *European Journal of Operational Research*, 230(3):539–550, nov 2013. doi: 10.1016/j.ejor.2013.05.001.

[30] Tan Fung Chan and R.V. Dubey. A weighted least-norm solution based scheme for avoiding joint limits for redundant joint manipulators. *IEEE Transactions on Robotics and Automation*, 11(2):286–292, apr 1995. doi: 10.1109/70.370511.

[31] Caroline P Carvalho Chanel, Charles Lesire, and Florent Teichteil-Königsbuch. A robotic execution framework for online probabilistic (re) planning. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*, 2014.

[32] Imran Ali Chaudhry and Abid Ali Khan. A research survey: review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, 23 (3):551–591, aug 2015. doi: 10.1111/itor.12199.

[33] Imran Ali Chaudhry and Abid Ali Khan. A research survey: review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, 23 (3):551–591, 2016. ISSN 1475-3995. doi: https://doi.org/10.1111/itor.12199. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12199.

[34] H. Chen, J. Ihlow, and C. Lehmann. A genetic algorithm for flexible job-shop schedul-

ing. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*. IEEE, 1999. doi: 10.1109/robot.1999.772512.

[35] S Chien, G Rabideau, R Knight, R Sherwood, B Engelhardt, D Mutz, T Estlin, B Smith, F Fisher, T Barrett, G Stebbins, and D Tran. ASPEN – Automated Planning and Scheduling for Space Mission Operations. page 10, 2000.

[36] Chetan Chudasama, SM Shah, and Mahesh Panchal. Comparison of parents selection methods of genetic algorithm for tsp. In *International Conference on Computer Communication and Networks CSI-COMNET-2011, Proceedings*, pages 85–87. Citeseer, 2011.

[37] Benjamin Cohen, Sachin Chitta, and Maxim Likhachev. Single- and dual-arm motion planning with heuristic search. *The International Journal of Robotics Research*, 33(2): 305–320, nov 2013. doi: 10.1177/0278364913507983.

[38] Elliott Coleshill, Layi Oshinowo, Richard Rembala, Bardia Bina, Daniel Rey, and Shelley Sindelar. Dextre: Improving maintenance operations on the international space station. *Acta Astronautica*, 64(9-10):869–874, may 2009. doi: 10.1016/j.actaastro.2008.11.011.

[39] Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. Numerical experiments with the lancelot package (release a) for large-scale nonlinear optimization. *Mathematical Programming*, 73:73–110, 1996.

[40] P.I. Corke. A simple and systematic approach to assigning denavit-hartenberg parameters. *IEEE Transactions on Robotics*, 23(3):590–594, jun 2007. doi: 10.1109/tro.2007.896765.

[41] Marco Antonio Cruz-Chávez, Martín G. Martínez-Rangel, and Martín H. Cruz-Rosales. Accelerated simulated annealing algorithm applied to the flexible job shop scheduling problem. *International Transactions in Operational Research*, 24(5):1119–1137, jul 2015. doi: 10.1111/itor.12195.

[42] Luis Cruz-Piris, Ivan Marsa-Maestre, and Miguel A. Lopez-Carmona. A variable-length chromosome genetic algorithm to solve a road traffic coordination multipath problem. *IEEE Access*, 7:111968–111981, 2019. doi: 10.1109/access.2019.2935041.

[43] Fantahun Defersha, Dolapo Obimuyiwa, and Alebachew D. Yimer. Multiple-trial/best-move simulated annealing for flexible job shop scheduling with scarce setup-operators. In *2020 7th International Conference on Soft Computing & Machine Intelligence (ISCMI)*. IEEE, nov 2020. doi: 10.1109/iscmi51676.2020.9311570.

[44] Ulrich Derigs. Using confidence limits for the global optimum in combinatorial optimization. *Operations Research*, 33(5):1024–1049, oct 1985. doi: 10.1287/opre.33.5.1024.

[45] Feng Ding and Cong Liu. Applying coordinate fixed denavit–hartenberg method to solve the workspace of drilling robot arm. *International Journal of Advanced Robotic Systems*, 15(4):172988141879328, jul 2018. doi: 10.1177/1729881418793283.

[46] Jun Dong and Jeffrey C. Trinkle. Orientation-based reachability map for robot base placement. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, sep 2015. doi: 10.1109/iros.2015.7353564.

[47] Yunfei Dong, Tianyu Ren, Ken Chen, and Dan Wu. An efficient robot payload identification method for industrial application. *Industrial Robot: An International Journal*, 45(4):505–515, jul 2018. doi: 10.1108/ir-03-2018-0037.

[48] Mehmet Ferlibas and Reza Ghabcheloo. Load weight estimation on an excavator in static and dynamic motions. In *Linköping Electronic Conference Proceedings*. Linköping University Electronic Press, jun 2021. doi: 10.3384/ecp182p90.

[49] Angelika Fetzner, Christian Frese, and Christian Frey. A 3d representation of obstacles in the robots reachable area considering occlusions. In *ISR/Robotik 2014; 41st International Symposium on Robotics*, pages 1–8. VDE, 2014.

[50] Angel Flores-Abad, Ou Ma, Khanh Pham, and Steve Ulrich. A review of space robotics technologies for on-orbit servicing. *Progress in Aerospace Sciences*, 68:1–26, jul 2014. doi: 10.1016/j.paerosci.2014.03.002.

[51] Eduardo Feo Flushing, Luca M. Gambardella, and Gianni A. Di Caro. Simultaneous task allocation, data routing, and transmission scheduling in mobile multi-robot teams. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, sep 2017. doi: 10.1109/iros.2017.8206002.

[52] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, dec 2003. doi: 10.1613/jair.1129.

[53] Khusniddin Fozilov, Yasuhisa Hasegawa, and Kosuke Sekiyama. Towards self-autonomy evaluation using behavior trees. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, oct 2021. doi: 10.1109/smc52423.2021.9658838.

[54] Bo Fu, William Smith, Denise Rizzo, Matthew Castanier, Maani Ghaffari, and Kira Barton. Robust task scheduling for heterogeneous robot teams under capability uncertainty, 2021.

[55] A. Fukunaga, G. Rabideau, S. Chien, and D. Yan. Towards an application framework for automated planning and scheduling. In *1997 IEEE Aerospace Conference*. IEEE, 1997. doi: 10.1109/aero.1997.574426.

[56] Eric Gillard. Nasa assemblers are putting the pieces together for in-space assembly, Dec 2019. URL https://tinyurl.com/26vn6a9v.

[57] Maria Gini. Multi-Robot Allocation of Tasks with Temporal and Ordering Constraints. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), February 2017. ISSN 2374-3468. URL https://ojs.aaai.org/index.php/AAAI/article/view/11145. Number: 1.

[58] Mauricio Granada-Echeverri, Eliana M. Toro, and Jhon Jairo Santa. A mixed integer linear programming formulation for the vehicle routing problem with backhauls. *International Journal of Industrial Engineering Computations*, pages 295–308, 2019. doi: 10.5267/j.ijiec.2018.6.003.

[59] Yisheng Guan and Kazuhito Yokoi. Reachable space generation of a humanoid robot using the monte carlo method. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, oct 2006. doi: 10.1109/iros.2006.282406.

[60] Deepti Gupta and Shabina Ghafir. An overview of methods maintaining diversity in genetic algorithms. *International journal of emerging technology and advanced engineering*, 2(5):56–60, 2012.

[61] LLC Gurobi Optimization. Gurobi optimizer reference manual. 2021. URL http://www.gurobi.com.

[62] Van-Phuong Ha, Trung-Kien Dao, Ngoc-Yen Pham, and Minh-Hoang Le. A variable-

length chromosome genetic algorithm for time-based sensor network schedule optimization. *Sensors*, 21(12):3990, jun 2021. doi: 10.3390/s21123990.

[63] A. Hanif Halim, I. Ismail, and Swagatam Das. Performance assessment of the metaheuristic optimization algorithms: an exhaustive review. *Artificial Intelligence Review*, 54(3):2323–2409, oct 2020. doi: 10.1007/s10462-020-09906-6.

[64] Limin Han, Graham Kendall, and Peter Cowling. AN ADAPTIVE LENGTH CHROMOSOME HYPER-HEURISTIC GENETIC ALGORITHM FOR a TRAINER SCHEDULING PROBLEM. In *Recent Advances in Simulated Evolution and Learning*, pages 506–525. WORLD SCIENTIFIC, aug 2004. doi: 10.1142/9789812561794_0027.

[65] Valentin Noah Hartmann, Andreas Orthey, Danny Driess, Ozgur S. Oguz, and Marc Toussaint. Long-horizon multi-robot rearrangement planning for construction assembly. 2021.

[66] Frederik W Heger and Sanjiv Singh. Robust robotic assembly through contingencies, plan repair and re-planning. In *2010 IEEE International Conference on Robotics and Automation*. IEEE, may 2010. doi: 10.1109/robot.2010.5509274.

[67] Xuewen Huang, , Xiaotong Zhang, Sardar M. N. Islam, Carlos A. Vega-Mejía, and and. An enhanced genetic algorithm with an innovative encoding strategy for flexible job-shop scheduling with operation and processing flexibility. *Journal of Industrial & Management Optimization*, 16(6):2943–2969, 2020. doi: 10.3934/jimo.2019088.

[68] Jacob Huckaby and Henrik Christensen. Modeling robot assembly tasks in manufacturing using sysml. In *ISR/Robotik 2014; 41st International Symposium on Robotics*, pages 1–7, 2014.

[69] Jacob O'Donnal Huckaby and Henrik I Christensen. A taxonomic framework for task

modeling and knowledge transfer in manufacturing robotics. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[70] Abid Hussain, Yousaf Shad Muhammad, M. Nauman Sajid, Ijaz Hussain, Alaa Mohamd Shoukry, and Showkat Gani. Genetic algorithm for traveling salesman problem with modified cycle crossover operator. *Computational Intelligence and Neuroscience*, 2017:1–7, 2017. doi: 10.1155/2017/7430125.

[71] Khalid Jebari and Mohammed Madiafi. Selection methods for genetic algorithms. *International Journal of Emerging Sciences*, 3(4):333–344, 2013.

[72] Jason Kalirai. Scientific discovery with the james webb space telescope. *Contemporary Physics*, 59(3):251–290, jul 2018. doi: 10.1080/00107514.2018.1467648.

[73] Sisir Karumanchi, Kyle Edelberg, Jeremy Nash, Charles Bergh, Russell Smith, Blair Emanuel, Jason Carlton, John Koehler, Junggon Kim, Rudranarayan Mukherjee, Brett Kennedy, and Paul Backes. Payload-centric autonomy for in-space robotic assembly of modular space structures. *Journal of Field Robotics*, 35(6):1005–1021, 2018. ISSN 1556-4967. doi: https://doi.org/10.1002/rob.21792. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21792.

[74] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80(5):8091–8126, oct 2020. doi: 10.1007/s11042-020-10139-6.

[75] Sungwoo Kim and Ilkyeong Moon. Traveling salesman problem with a drone station. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(1):42–52, jan 2019. doi: 10.1109/tsmc.2018.2867496.

[76] Mitsuhiko Kimoto, Yuuki Yasumatsu, and Michita Imai. Help-estimator: Robot requests for help from humans by estimating a person's subjective time. *International Journal of Social Robotics*, jul 2021. doi: 10.1007/s12369-021-00792-8.

[77] Dan King. Space servicing: past, present and future. In *Proceedings of the 6th International Symposium on Artificial Intelligence and Robotics & Automation in Space: i-SAIRAS*, pages 18–22, 2001.

[78] Tamás Kis. Job-shop scheduling with processing alternatives. *European Journal of Operational Research*, 151(2):307–332, dec 2003. doi: 10.1016/s0377-2217(02)00828-7.

[79] Ross A. Knepper, Todd Layton, John Romanishin, and Daniela Rus. IkeaBot: An autonomous multi-robot coordinated furniture assembly system. In *2013 IEEE International Conference on Robotics and Automation*. IEEE, may 2013. doi: 10.1109/icra.2013.6630673.

[80] S. Knight, G. Rabideau, S. Chien, B. Engelhardt, and R. Sherwood. Casper: space exploration through continuous planning. *IEEE Intelligent Systems*, 16(5):70–75, September 2001. doi: 10.1109/MIS.2001.956084.

[81] Konstantinos Kokkalis, George Michalos, Panagiotis Aivaliotis, and Sotiris Makris. An approach for implementing power and force limiting in sensorless industrial robots. *Procedia CIRP*, 76:138–143, 2018. doi: 10.1016/j.procir.2018.01.028.

[82] G. Ayorkor Korsah, Balajee Kannan, Brett Browning, Anthony Stentz, and M. Bernardine Dias. xBots: An approach to generating and executing optimal multi-robot plans with cross-schedule dependencies. In *2012 IEEE International Conference on Robotics and Automation*, pages 115–122, May 2012. doi: 10.1109/ICRA.2012.6225234. ISSN: 1050-4729.

[83] Edoardo Lamon, Alessandro De Franco, Luka Peternel, and Arash Ajoudani. A capability-aware role allocation approach to industrial assembly tasks. *IEEE Robotics and Automation Letters*, 4(4):3378–3385, oct 2019. doi: 10.1109/lra.2019.2926963.

[84] C-Y Lee and EK Antonsson. Variable length genomes for evolutionary algorithms. In *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, pages 806–806, 2000.

[85] Nicolas Lee, Paul Backes, Joel Burdick, Sergio Pellegrino, Christine Fuller, Kristina Hogstrom, Brett Kennedy, Junggon Kim, Rudranarayan Mukherjee, Carl Seubert, and Yen-Hung Wu. Architecture for in-space robotic assembly of a modular space telescope. *Journal of Astronomical Telescopes, Instruments, and Systems*, 2(4):041207, jul 2016. doi: 10.1117/1.jatis.2.4.041207.

[86] Daria Leiber, Veit Hammerstingl, Felix Weiß, and Gunter Reinhart. Automated design of multi-station assembly lines. *Procedia CIRP*, 79:137–142, 2019. doi: 10.1016/j.procir.2019.02.029.

[87] Jia Li, Fei Zhao, Xingchen Li, and Junjie Li. Analysis of robotic workspace based on monte carlo method and the posture matrix. In *2016 IEEE International Conference on Control and Robotics Engineering (ICCRE)*. IEEE, apr 2016. doi: 10.1109/iccre.2016.7476145.

[88] Xinyu Li and Liang Gao. *Effective methods for integrated process planning and scheduling*, volume 2. Springer, 2020.

[89] Yee Yeng Liau and Kwangyeol Ryu. Task allocation in human-robot collaboration (HRC) based on task characteristics and agent capability for mold assembly. *Procedia Manufacturing*, 51:179–186, 2020. doi: 10.1016/j.promfg.2020.10.026.

[90] Siew Mooi Lim, Abu Bakar Md Sultan, Md Nasir Sulaiman, Aida Mustapha, and Kuan Yew Leong. Crossover and mutation operators of genetic algorithms. *International journal of machine learning and computing*, 7(1):9–12, 2017.

[91] Martina Lippi and Alessandro Marino. A mixed-integer linear programming formulation for human multi-robot task allocation. In *2021 30th IEEE International Conference on Robot &amp Human Interactive Communication (RO-MAN)*. IEEE, aug 2021. doi: 10.1109/ro-man50785.2021.9515362.

[92] Qihao Liu, Xinyu Li, Liang Gao, and Yingli Li. A modified genetic algorithm with new encoding and decoding methods for integrated process planning and scheduling problem. *IEEE Transactions on Cybernetics*, 51(9):4429–4438, sep 2021. doi: 10.1109/tcyb.2020.3026651.

[93] Michele Lombardi and Michela Milano. Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. *Constraints*, 17(1):51–85, jan 2012. doi: 10.1007/s10601-011-9115-6.

[94] Kevin M Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge Univeristy Press, 2017. ISBN 978-1107156302.

[95] Abhijit Makhal and Alex K. Goins. Reuleaux: Robot base placement by reachability analysis. In *2018 Second IEEE International Conference on Robotic Computing (IRC)*. IEEE, jan 2018. doi: 10.1109/irc.2018.00028.

[96] Ilias El Makrini, Kelly Merckaert, Joris De Winter, Dirk Lefeber, and Bram Vanderborght. Task allocation for improved ergonomics in human-robot collaborative assembly. *Interaction Studies*, 20(1):102–133, jul 2019. doi: 10.1075/is.18018.mak.

[97] Jeremy A. Marvel, Roger Bostelman, and Joe Falco. Multi-robot assembly strategies and metrics. *ACM Computing Surveys*, 51(1):1–32, jan 2019. doi: 10.1145/3150225.

[98] Michael L Mauldin et al. Maintaining diversity in genetic search. In *AAAI*, pages 247–250, 1984.

[99] Conor McGann, Frederic Py, Kanna Rajan, Hans Thomas, Richard Henthorn, and Rob McEwen. A deliberative architecture for AUV control. In *2008 IEEE International Conference on Robotics and Automation*. IEEE, may 2008. doi: 10.1109/robot.2008.4543343.

[100] Joshua Moser, Melanie Anderson, Holly Everson, Amy Quartaro, William D. Chapin, Benjamin Beach, Julia Hoffman, Robert Hildebrand, and Erik E. Komendera. Recent developments in robust, accurate autonomous assembly methods for surface and orbital structures. In *ASCEND 2020*. American Institute of Aeronautics and Astronautics, nov 2020. doi: 10.2514/6.2020-4270.

[101] Joshua Moser, Julia Hoffman, Robert Hildebrand, and Erik Komendera. An autonomous task assignment paradigm for autonomous robotic in-space assembly. *Frontiers in Robotics and AI*, 9, feb 2022. doi: 10.3389/frobt.2022.709905.

[102] R Mukherjee, N Siegler, and H Thronson. The future of space astronomy will be built: results from the in-space astronomical telescope (isat) assembly design study. In *70th International Astronautical Congress (IAC)*, pages 21–25, 2019.

[103] T. Murata and H. Ishibuchi. Positive and negative combination effects of crossover and mutation operators in sequencing problems. In *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE. doi: 10.1109/icec.1996.542355.

[104] Angadh Nanjangud, Craig I Underwood, Christopher P Bridges, M. Saaj

Chakravarthini, Steve Eckersley, Martin Sweeting, and Paolo Biancod. Towards robotic on-orbit assembly of large space telescopes: Mission architectures, concepts, and analyses. In *Proceedings of the International Astronautical Congress, IAC*. International Astronautical Federation, 2019.

[105] Jianjun Ni, Kang Wang, Haohao Huang, Liuying Wu, and Chengming Luo. Robot path planning based on an improved genetic algorithm with variable length chromosome. In *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. IEEE, aug 2016. doi: 10.1109/fskd.2016.7603165.

[106] Nikolaos Nikolakis, Niki Kousi, George Michalos, and Sotiris Makris. Dynamic scheduling of shared human-robot manufacturing operations. *Procedia CIRP*, 72:9–14, 2018. doi: 10.1016/j.procir.2018.04.007.

[107] Amir Hossein Nobil, Seyed Mohammad Ebrahim Sharifnia, and Leopoldo Eduardo Cárdenas-Barrón. Mixed integer linear programming problem for personnel multi-day shift scheduling: A case study in an iran hospital. *Alexandria Engineering Journal*, 61 (1):419–426, jan 2022. doi: 10.1016/j.aej.2021.06.030.

[108] Ernesto Nunes, Marie Manner, Hakim Mitiche, and Maria Gini. A taxonomy for task allocation problems with temporal and ordering constraints. *Robotics and Autonomous Systems*, 90:55–70, apr 2017. doi: 10.1016/j.robot.2016.10.008.

[109] I. Ono, M. Yamamura, and S. Kobayashi. A genetic algorithm for job-shop scheduling problems using job-based order crossover. In *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE. doi: 10.1109/icec.1996.542658.

[110] F. Pezzella, G. Morganti, and G. Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202–3212, oct 2008. doi: 10.1016/j.cor.2007.02.014.

[111] Julius Pfrommer, Miriam Schleipen, and Jurgen Beyerer. PPRS: Production skills and their relation to product, process, and resource. In *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, sep 2013. doi: 10.1109/etfa.2013.6648114.

[112] Petrica C. Pop. New integer programming formulations of the generalized travelling salesman problem. *American Journal of Applied Sciences*, 4(11):932–937, nov 2007. doi: 10.3844/ajassp.2007.932.937.

[113] Amir Rajabinasab and Saeed Mansour. Dynamic flexible job shop scheduling with alternative process plans: an agent-based approach. *The International Journal of Advanced Manufacturing Technology*, 54(9-12):1091–1107, oct 2010. doi: 10.1007/s00170-010-2986-7.

[114] Kanna Rajan, Frédéric Py, and Javier Barreiro. Towards Deliberative Control in Marine Robotics. In Mae L. Seto, editor, *Marine Robot Autonomy*, pages 91–175. Springer New York, New York, NY, 2013. ISBN 978-1-4614-5659-9. doi: 10.1007/978-1-4614-5659-9_3.

[115] Fabian Ranz, Vera Hummel, and Wilfried Sihn. Capability-based task allocation in human-robot collaboration. *Procedia Manufacturing*, 9:182–189, 2017. doi: 10.1016/j.promfg.2017.04.011.

[116] Benjamin Riviere, Wolfgang Honig, Yisong Yue, and Soon-Jo Chung. GLAS: Global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning. *IEEE Robotics and Automation Letters*, 5(3):4249–4256, jul 2020. doi: 10.1109/lra.2020.2994035.

[117] Ismael Rodriguez, Adrian S. Bauer, Korbinian Nottensteiner, Daniel Leidner, Gerhard Grunwald, and Maximo A. Roa. Autonomous robot planning system for in-space

assembly of reconfigurable structures. In *2021 IEEE Aerospace Conference (50100)*. IEEE, mar 2021. doi: 10.1109/aero50100.2021.9438257.

[118] Mathieu Rognant, Christelle Cumer, Jean-Marc Biannic, Maximo A. Roa, Antoine Verhaeghe, and Vincent Bissonnette. Autonomous assembly of large structures in space: a technology review. In *EUCASS 2019*, Madrid, Spain, July 2019. URL https://hal.archives-ouvertes.fr/hal-03316188.

[119] Michael Roth, Carsten Fritsche, Gustaf Hendeby, and Fredrik Gustafison. The ensemble kalman filter and its relations to other nonlinear filters. In *2015 23rd European Signal Processing Conference (EUSIPCO)*. IEEE, aug 2015. doi: 10.1109/eusipco.2015.7362581.

[120] Mohammad Saidi-Mehrabad and Parviz Fattahi. Flexible job shop scheduling with tabu search algorithms. *The International Journal of Advanced Manufacturing Technology*, 32(5-6):563–570, may 2006. doi: 10.1007/s00170-005-0375-4.

[121] Chris Saunders, Dan Lobb, Martin Sweeting, and Yang Gao. Building large telescopes in orbit using small satellites. *Acta Astronautica*, 141:183–195, dec 2017. doi: 10.1016/j.actaastro.2017.09.022.

[122] Nayeli Jazmin Escamilla Serna, Juan Carlos Seck-Tuoh-Mora, Joselito Medina-Marin, Norberto Hernandez-Romero, Irving Barragan-Vite, and Jose Ramon Corona Armenta. A global-local neighborhood search algorithm and tabu search for flexible job shop scheduling problem. *PeerJ Computer Science*, 7:e574, may 2021. doi: 10.7717/peerj-cs.574.

[123] Xinyu Shao, Xinyu Li, Liang Gao, and Chaoyong Zhang. Integration of process planning and scheduling—a modified genetic algorithm-based approach. *Computers & Operations Research*, 36(6):2082–2096, jun 2009. doi: 10.1016/j.cor.2008.07.006.

[124] Anupriya Shukla, Hari Mohan Pandey, and Deepti Mehrotra. Comparative review of selection techniques in genetic algorithm. In *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*. IEEE, feb 2015. doi: 10.1109/ablaze.2015.7154916.

[125] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Springer, 2016.

[126] Nick Siegler. Building the future: in-space servicing and assembly of large aperture space telescopes. 2018.

[127] V. Singh and S. Choudhary. Genetic algorithm for traveling salesman problem: Using modified partially-mapped crossover operator. In *2009 International Multimedia, Signal Processing and Communication Technologies*. IEEE, mar 2009. doi: 10.1109/mspct.2009.5164164.

[128] Marshall Smith, Douglas Craig, Nicole Herrmann, Erin Mahoney, Jonathan Krezel, Nate McIntyre, and Kandyce Goodliff. The artemis program: An overview of nasa's activities to return humans to the moon. In *2020 IEEE Aerospace Conference*, pages 1–10. IEEE, 2020.

[129] Roman G Strongin and Yaroslav D Sergeyev. *Global optimization with non-convex constraints: Sequential and parallel algorithms*, volume 45. Springer Science & Business Media, 2013.

[130] June sup Yi, Min Sung Ahn, Hosik Chae, Hyunwoo Nam, Donghun Noh, Dennis Hong, and Hyungpil Moon. Task planning with mixed-integer programming for multiple cooking task using dual-arm robot. In *2020 17th International Conference on Ubiquitous Robots (UR)*. IEEE, jun 2020. doi: 10.1109/ur49135.2020.9144803.

[131] Jekan Thangavelautham, Aman Chandra, and Erik Jensen. Autonomous Robot Teams

for Lunar Mining Base Construction and Operation. In *2020 IEEE Aerospace Conference*, pages 1–16, March 2020. doi: 10.1109/AERO47225.2020.9172811. ISSN: 1095-323X.

[132] Panagiota Tsarouchi, George Michalos, Sotiris Makris, Thanasis Athanasatos, Konstantinos Dimoulas, and George Chryssolouris. On a human–robot workplace design and task allocation system. *International Journal of Computer Integrated Manufacturing*, 30(12):1272–1279, apr 2017. doi: 10.1080/0951192x.2017.1307524.

[133] Anant J Umbarkar and Pranali D Sheth. Crossover operators in genetic algorithms: a review. *ICTACT journal on soft computing*, 6(1), 2015.

[134] Nikolaus Vahrenkamp, Tamim Asfour, Giorgio Metta, Giulio Sandini, and Rudiger Dillmann. Manipulability analysis. In *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. IEEE, nov 2012. doi: 10.1109/humanoids.2012.6651576.

[135] Nikolaus Vahrenkamp, Harry Arnst, Mirko Wachter, David Schiebener, Panagiotis Sotiropoulos, Michal Kowalik, and Tamim Asfour. Workspace analysis for planning human-robot interaction tasks. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. IEEE, nov 2016. doi: 10.1109/humanoids.2016.7803437.

[136] Vijay Kumar Verma and Biresh Kumar. Genetic algorithm: an overview and its application. *International Journal of Advanced Studies in Computers, Science and Engineering*, 3(2):21, 2014.

[137] Jun Wan, HongTao Wu, Rui Ma, and Liang'an Zhang. A study on avoiding joint limits for inverse kinematics of redundant manipulators using improved clamping weighted

least-norm method. *Journal of Mechanical Science and Technology*, 32(3):1367–1378, mar 2018. doi: 10.1007/s12206-018-0240-7.

[138] Zheng Wang and Jiuh-Biing Sheu. Vehicle routing problem with drones. *Transportation Research Part B: Methodological*, 122:350–364, apr 2019. doi: 10.1016/j.trb.2019.03.005.

[139] Markus Wilde, Brian Kaplinger, Tiauw Go, Hector Gutierrez, and Daniel Kirk. ORION: A simulation environment for spacecraft formation flight, capture, and orbital robotics. In *2016 IEEE Aerospace Conference*. IEEE, mar 2016. doi: 10.1109/aero.2016.7500575.

[140] Cuebong Wong, Erfu Yang, Xiu-Tian Yan, and Dongbing Gu. Autonomous robots for harsh environments: a holistic overview of current solutions and ongoing challenges. *Systems Science & Control Engineering*, 6(1):213–219, jan 2018. doi: 10.1080/21642583.2018.1477634.

[141] Aixuan Wu, Zhiping Shi, Yongdong Li, Minhua Wu, Yong Guan, Jie Zhang, and Hongxing Wei. Formal kinematic analysis of a general 6r manipulator using the screw theory. *Mathematical Problems in Engineering*, 2015:1–7, 2015. doi: 10.1155/2015/549797.

[142] Zhihui XUE, Jinguo LIU, Chenchen WU, and Yuchuang TONG. Review of in-space assembly technologies. *Chinese Journal of Aeronautics*, 34(11):21–47, nov 2021. doi: 10.1016/j.cja.2020.09.043.

[143] Hao Yang, Tavan Eftekhar, Chad Esselink, Yan Ding, and Shiqi Zhang. Task and situation structures for case-based planning. In Antonio A. Sánchez-Ruiz and Michael W. Floyd, editors, *Case-Based Reasoning Research and Development*, pages 263–278, Cham, 2021. Springer International Publishing. ISBN 978-3-030-86957-1.

[144] Xin-She Yang. Genetic algorithms. In *Nature-Inspired Optimization Algorithms*, pages 91–100. Elsevier, 2021. doi: 10.1016/b978-0-12-821986-7.00013-5.

[145] Yuan Yuan, Diego Cattaruzza, Maxime Ogier, Cyriaque Rousselot, and Frédéric Semet. Mixed integer programming formulations for the generalized traveling salesman problem with time windows. *4OR*, 19(4):571–592, oct 2020. doi: 10.1007/s10288-020-00461-y.

[146] Franziska Zacharias, Christoph Borst, and Gerd Hirzinger. Capturing robot workspace structure: representing robot capabilities. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, oct 2007. doi: 10.1109/iros.2007.4399105.

[147] FRANZISKA ZACHARIAS, CHRISTOPH BORST, SEBASTIAN WOLF, and GERD HIRZINGER. THE CAPABILITY MAP: A TOOL TO ANALYZE ROBOT ARM WORKSPACES. *International Journal of Humanoid Robotics*, 10(04):1350031, dec 2013. doi: 10.1142/s021984361350031x.

[148] Fan Zeng, Juliang Xiao, and Haitao Liu. Force/torque sensorless compliant control strategy for assembly tasks using a 6-DOF collaborative robot. *IEEE Access*, 7:108795–108805, 2019. doi: 10.1109/access.2019.2931515.

[149] Luping Zhang and T.N. Wong. An object-coding genetic algorithm for integrated process planning and scheduling. *European Journal of Operational Research*, 244(2):434–444, jul 2015. doi: 10.1016/j.ejor.2015.01.032.

[150] Cemal Özgüven, Lale Özbakır, and Yasemin Yavuz. Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling*, 34(6):1539–1548, jun 2010. ISSN 0307-904X. doi: 10.1016/j.apm.2009.09.002. URL https://www.sciencedirect.com/science/article/pii/S0307904X09002819.

[151] Cemal Özgüven, Lale Özbakır, and Yasemin Yavuz. Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling*, 34(6):1539–1548, June 2010. ISSN 0307-904X. doi: 10.1016/j.apm.2009.09.002. URL https://www.sciencedirect.com/science/article/pii/S0307904X09002819.

Appendices

Appendix A

Arch Assembly Project Definition

A.1 Pose Definitions

Table A.1: SAPD pose definitions for the arch assembly project.

Pose A (W_A)	**Pose B** (W_B)	**Pose C** (W_C)	**Pose D** (W_D)
Column component storage area	Horizontal crossmember components storage area	Column component staging area	Horizontal crossmember components staging area
Properties: • reference • (70, 185)	Properties: • reference • (70, 155)	Properties: • reference • (70, 115)	Properties: • reference • (70, 85)
Pose E (W_E)	**Pose Sr1** (W_{Sr1})	**Pose Sr2** (W_{Sr2})	—
Final Assembly area	MARC 1 starting location	MARC 2 starting location	—
Properties: • reference • (70, 40)	Properties: • reference • (20, 185)	Properties: • reference • (20, 140)	—

A.2 Component Definitions

Table A.2: SAPD component definitions for the arch assembly project.

Component lb1 (C_{lb1})	**Component lb2** (C_{lb2})	**Component mb1** (W_{mb1})	**Component mb2** (W_{mb2})
First large block	Second large block	First medium block	Second medium block
Properties: • Large block • Start: W_A	Properties: • Large block • Start: W_A	Properties: • Medium block • Start: W_A	Properties: • Medium block • Start: W_A
State: • Not in position • Not broken • current: (70, 185)	State: • Not in position • Not broken • current: (70, 185)	State: • Not in position • Not broken • current: (70, 185)	State: • Not in position • Not broken • current: (70, 185)
Component sbc1 (C_{sbc1})	**Component sbc2** (C_{sbc2})	**Component sbm1** (C_{sbm1})	—
First small corner block	Second small corner block	Small middle block	—
Properties: • Small corner block • Start: W_B	Properties: • Small corner block • Start: W_B	Properties: • Small middle block • Start: W_B	—
State: • Not in position • Not broken • current: (70, 155)	State: • Not in position • Not broken • current: (70, 155)	State: • Not in position • Not broken • current: (70, 155)	—

A.3 Joint Definitions

Table A.3: SAPD joint definitions for the arch assembly project.

Connection 1 (K_1)	**Connection 2** (K_2)	**Connection 3** (K_3)
Connection between lb1 and mb1 to form the first column	Connection between lb2 and mb2 to form the second column	Connection between sbc1 and sbm1 to form part of the crossmember
Properties: • Clip • $[C_{lb1},\ C_{mb1}]$	Properties: • Clip • $[C_{lb2},\ C_{mb2}]$	Properties: • Clip • $[C_{sbc1},\ C_{sbm1}]$
State: • Not joined	State: • Not joined	State: • Not joined
Connection 4 (K_4)	**Connection 5** (K_5)	**Connection 6** (K_6)
Connection between sbc2 and sbm1 to form part of the crossmember	Connection between sbc1 and mb1 to connect the crossmember to the first column	Connection between sbc2 and mb2 to connect the crossmember to the second column
Properties: • Clip • $[C_{sbc2},\ C_{sbm1}]$	Properties: • Clip • $[C_{sbc1},\ C_{mb1}]$	Properties: • Clip • $[C_{sbc2},\ C_{mb2}]$
State: • Not joined	State: • Not joined	State: • Not joined

A.4 Robot Definitions

Table A.4: SAPD robot definitions for the arch assembly problem.

MARC 1 (R_{m1})	**MARC 2** (R_{m2})
State: • idle • current: (20, 185) • task: none	State: • idle • current: (20, 140) • task: none
Properties: • Gripper • Mobile: Yes • Abilities: – Idle: 0.01 (s) – Locomote: 6.64 (in/s) – Connect component: 94.33 (s) – Connect subassembly: 90.1 (s) – Co-op connect component: 165.11 (s) – Co-op connect subassembly: 79.58 (s) – Pick & Place: 41.67 (s) – Place: 15.61 (s)	Properties: • Gripper • Mobile: Yes • Abilities: – Idle: 0.01 (s) – Locomote: 6.16 (in/s) – Connect component: 106.14 (s) – Connect subassembly: 86.14 (s) – Co-op connect component: 165.11 (s) – Co-op connect subassembly: 79.58 (s) – Pick & Place: 66.72 (s) – Place: 20.56 (s)

A.5 Job and Process Plan Definitions

Table A.5: Jobs in the arch assembly project.

Job	Type	Component(s)	Location(s)	Description
Sr1	Start	N/A	W_{sr1}	Used to ensure MARC 1 starts at the right location
Sr2	Start	N/A	W_{sr2}	Used to ensure MARC 2 starts at the right location
Mlb1	Move	lb1	$W_A \rightarrow W_C$	Move the first large block from storage to staging
Mlb2	Move	lb2	$W_A \rightarrow W_C$	Move the second large block from storage to staging
Mmb1	Move	mb1	$W_A \rightarrow W_C$	Move the first medium block from storage to staging
Mmb2	Move	mb2	$W_A \rightarrow W_C$	Move the second medium block from storage to staging
Msbc1	Move	sbc1	$W_B \rightarrow W_D$	Move the first small corner block from storage to staging
Msbc2	Move	sbc2	$W_B \rightarrow W_D$	Moving the second small corner block from storage to staging
Msbm1	Move	sbm1	$W_B \rightarrow W_D$	Moving the small medium block from storage to staging
Cmb1lb1	Connect C	mb1 & lb1	W_C	Connecting the first large and medium blocks to make the first column
Cmb2lb2	Connect C	mb2 & lb2	W_C	Connecting the second large and medium blocks to make the second column
Mmb1lb1	Connect C	mb1 & lb1	$W_C \rightarrow W_E$	Move the first column from staging to the final assembly location
Mmb2lb2	Move	mb2 & lb2	$W_C \rightarrow W_E$	Move the second column from staging to the final assembly location
Csbm1sbc1	Connect C	smb1 & sbc1	W_D	Connecting the small middle block with the first small corner block to make part of the horizontal crossmember
Csbc2sbm1	Connect C	sbc2 & sbm1	W_D	Connecting the small middle block with the second small corner block to make part of the horizontal crossmember
Msbc1sbm1sbc2	Move	sbc1 & sbm1 & sbc2	$W_D \rightarrow W_E$	Move the crossmember to the final assembly location
Csbc1sbm1sbc2	Connect S	sbc1 & sbm2 & sbc2	W_E	Connect both sides of the crossmember to the two columns

Table A.6: Process plans and operations for each of the types of jobs.

Job Type	**Process plan(s)**	**Operation(s)**
Start	p0	[Idle]
Move	p0	[Pick & Place]
Connect C	p0	[Connect component]
	p1	[Co-op connect component, Co-op connect component]
Connect S	p0	[Connect subassembly]
	p1	[Co-op connect subassembly, Co-op connect subassembly]

Appendix B

Truss Block Wall Assembly Project Definition

B.1 Pose Definitions

Table B.1: SAPD pose definitions for the three by two truss block assembly project.

Pose Srn (W_{rn})	**Pose sb** (W_{sb})	**Pose 1** (W_1)	**Pose 2** (W_2)
Starting positing for robot n	Storage position for the blocks	First pose along the wall	Second pose along the wall
Properties: • reference • (0, 60, 0)	Properties: • reference • (100, 0, 0)	Properties: • reference • (100, 60, 0)	Properties: • reference • (100, 70, 0)
Pose 3 (W_3)	**Pose 4** (W_4)	**Pose 5** (W_5)	**Pose 6** (W_6)
Third pose along the wall	Fourth pose along the wall	Fifth pose along the wall	Sixth pose along the wall
Properties: • reference • (100, 80, 0)	Properties: • reference • (100, 90, 0)	Properties: • reference • (100, 100, 0)	Properties: • reference • (100, 110, 0)

B.2 Component Definitions

Table B.2: SAPD component definitions for the three by two truss block assembly project.

Component B21 (C_{B21})	**Component B22** (C_{B22})	**Component B23** (W_{B23})	**Component B24** (W_{B24})
First block in 2nd row	Second block in 2nd row	Third block in 2nd row	Fourth block in 2nd row
Properties: • block • Start: W_{sb}	Properties: • block • Start: W_{sb}	Properties: • block • Start: W_{sb}	Properties: • block • Start: W_{sb}
State: • Not in position • Not broken • current: W_{sb}	State: • Not in position • Not broken • current:W_{sb}	State: • Not in position • Not broken • current: W_{sb}	State: • Not in position • Not broken • current: W_{sb}
Component B11 (C_{B11})	**Component B12** (C_{B12})	**Component B13** (C_{B13})	—
First block in 1st row	Second block in 1st row	Third block in 1st row	—
Properties: • block • Start: W_{sb}	Properties: • block • Start: W_{sb}	Properties: • block • Start: W_{sb}	—
State: • Not in position • Not broken • current: W_{sb}	State: • Not in position • Not broken • current: W_{sb}	State: • Not in position • Not broken • current: W_{sb}	—

B.3 Joint Definitions

Table B.3: SAPD joint definitions for the three by two truss assembly project.

Connection 1 (K_1)	**Connection 2** (K_2)	**Connection 3** (K_3)	**Connection 4** (K_4)
Connection between B11 and the anchor point	Connection between B11 and B12	Connection between B12 and B13	Connection between B11 and B21
Properties: • Weld • $[C_{B11}]$	Properties: • Weld • $[C_{B11},\ C_{B12}]$	Properties: • Weld • $[C_{B12},\ C_{B13}]$	Properties: • Weld • $[C_{B11},\ C_{B21}]$
State: • Not joined	State: • Not joined	State: • Not joined	State: • Not joined
Connection 5 (K_5)	**Connection 6** (K_6)	**Connection 7** (K_7)	—
Connection between B21 and B22	Connection between B22 and B23	Connection between B23 and B24	—
Properties: • Weld • $[C_{sbc2},\ C_{sbm1}]$	Properties: • Weld • $[C_{sbc1},\ C_{mb1}]$	Properties: • Weld • $[C_{sbc1},\ C_{mb1}]$	— —
State: • Not joined	State: • Not joined	State: • Not joined	— —

B.4 Robot Definitions

Table B.4: SAPD robot team 1 definitions for the truss wall assembly problem.

Robot 1 (R_{m1})	**Robot 2** (R_{m2})
State: • idle • current: W_{sr1} • task: none	State: • idle • current: W_{sr2} • task: none
Properties: • Gripper • Mobile: Yes • Abilities: – Idle: $\mathcal{N}(0.01, 0.0001)$ (s) – Locomote: $\mathcal{N}(15.0, 2)$ (unit/s) – Grasp: $\mathcal{N}(10.0, 2)$ (s) – Connect: $\mathcal{N}(1.0E^4, 10)$ (s)	Properties: • Welder • Mobile: Yes • Abilities: – Idle: $\mathcal{N}(0.01, 0.0001)$ (s) – Locomote: $\mathcal{N}(15.0, 2)$ (in/s) – Grasp: $\mathcal{N}(30.0, 2)$ (s) – Connect: $\mathcal{N}(5.0, 1)$ (s)

Table B.5: SAPD robot team 2 definitions for the truss wall assembly problem.

Robot 3(R_{m3})	**Robot 4** (R_{m4})
State: • idle • current: W_{sr3} • task: none	State: • idle • current: W_{sr4} • task: none
Properties: • Gripper • Mobile: Yes • Abilities: – Idle: $\mathcal{N}(0.01, 0.0001)$ (s) – Locomote: $\mathcal{N}(20.0, 2)$ (unit/s) – Grasp: $\mathcal{N}(10.0, 2)$ (s) – Connect: $\mathcal{N}(1.0E^4, 10)$ (s)	Properties: • Welder • Mobile: Yes • Abilities: – Idle: $\mathcal{N}(0.01, 0.0001)$ (s) – Locomote: $\mathcal{N}(20.0, 2)$ (in/s) – Grasp: $\mathcal{N}(30.0, 2)$ (s) – Connect: $\mathcal{N}(5.0, 1)$ (s)

B.5 Job and Process Plan Definitions

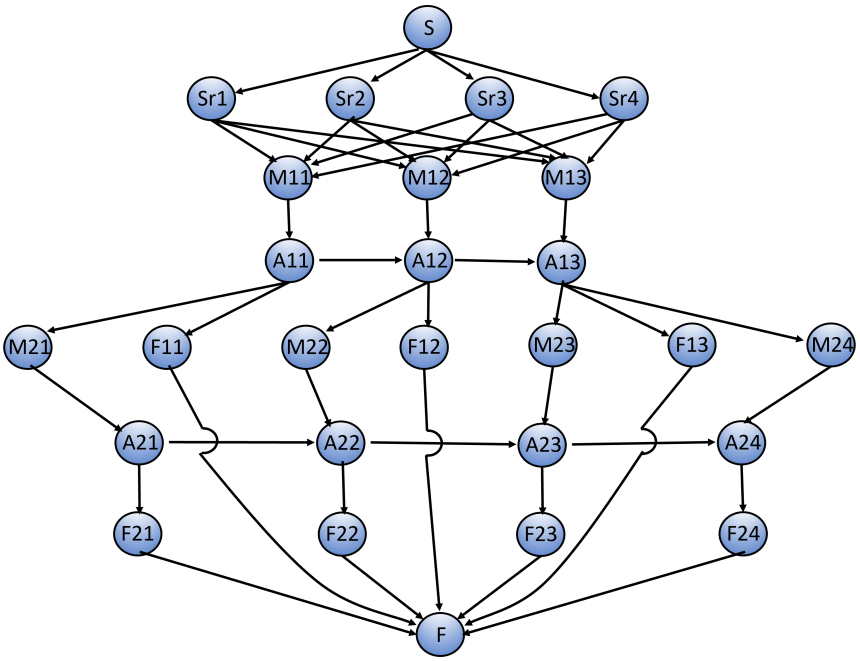


Figure B.1: Precedence constraint set for the three by two truss block wall assembly problem instance.

Jobs	Process Plans	Operations	Valid Machines	Continuity
Sn	P0	O1: idle	Rn	A11 : O1 → F12 : O1
Mrb	P0	O1: grasp	R1 R2 R3 R4	A12 : O1 → F12 : O1
Arb	P0	O1: grasp	R1 R2 R3 R4	A13 : O1 → F13 : O1
Frb	P0	O1: grasp O2: connect	R1 R2 R3 R4	A21 : O1 → F21 : O1
				A22 : O1 → F22 : O1
				A23 : O1 → F23 : O1
				A24 : O1 → F24 : O1

Figure B.2: Operations and continuity constraints for the three by two truss block wall assembly project instance.

Table B.6: Jobs in the three by two truss block wall assembly project.

Job	**Type**	**Component(s)**	**Location(s)**	**Description**
Sr1	Start	N/A	W_{sr1}	Used to ensure Robot 1 starts at the right location
Sr2	Start	N/A	W_{sr2}	Used to ensure Robot 2 starts at the right location
Sr3	Start	N/A	W_{sr3}	Used to ensure Robot 3 starts at the right location
Sr4	Start	N/A	W_{sr4}	Used to ensure Robot 4 starts at the right location
M11	Move	B11	$W_{sb} \rightarrow W_2$	Move the block from storage
M12	Move	B12	$W_{sb} \rightarrow W_4$	Move the block from storage
M13	Move	B13	$W_{sb} \rightarrow W_6$	Move the block from storage
A11	Align	B11	W_2	Align the block
A12	Align	B12	W_4	Align the block
A13	Align	B13	W_6	Align the block
F11	Fasten	B11	W_2	Fasten the block into place
F12	Fasten	B12	W_4	Fasten the block into place
F13	Fasten	B13	W_6	Fasten the block into place
M21	Move	B21	$W_{sb} \rightarrow W_1$	Move the block from storage
M22	Move	B22	$W_{sb} \rightarrow W_3$	Move the block from storage
M23	Move	B23	$W_{sb} \rightarrow W_5$	Move the block from storage
M24	Move	B24	$W_{sb} \rightarrow W_6$	Move the block from storage
A21	Align	B21	W_6	Align the block
A22	Align	B22	W_6	Align the block
A23	Align	B23	W_6	Align the block
A24	Align	B24	W_6	Align the block
F21	Fasten	B21	W_1	Fasten the block into place
F22	Fasten	B22	W_3	Fasten the block into place
F23	Fasten	B23	W_5	Fasten the block into place
F24	Fasten	B24	W_6	Fasten the block into place

Appendix C

GA and MIP Schedule Comparison

C.1 Arch Assembly Makespan Distributions

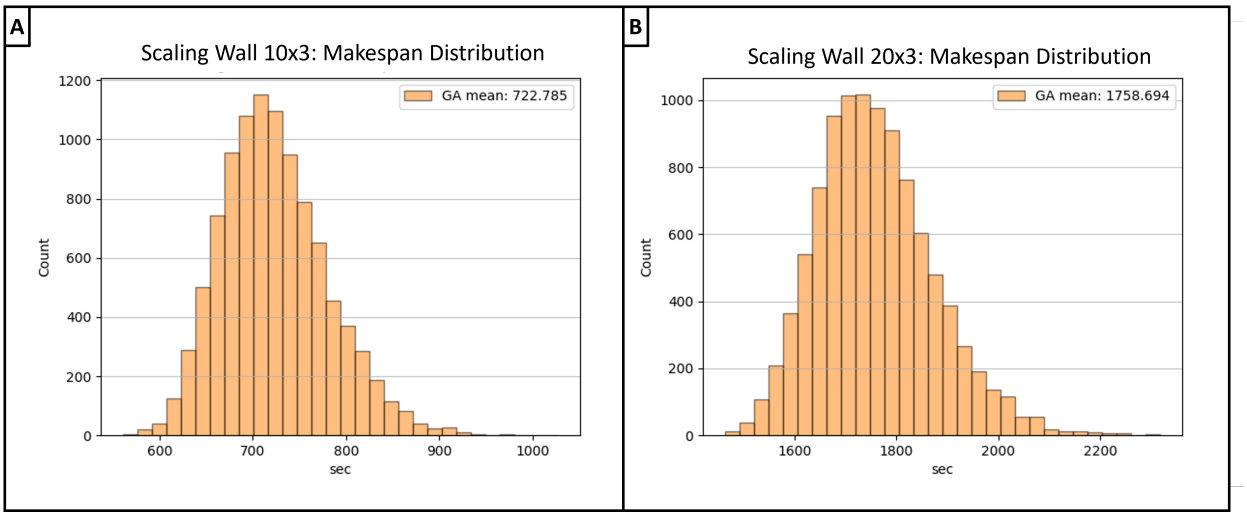


Figure C.1: Panel A: Scaling wall 10 x 3 makespan distribution for GA (the MIP could not find a valid solution) using 10,000 processing time samples. Panel B: Scaling wall 20 x 3 makespan distribution for GA (the MIP could not find a valid solution) using 10,000 processing time samples.

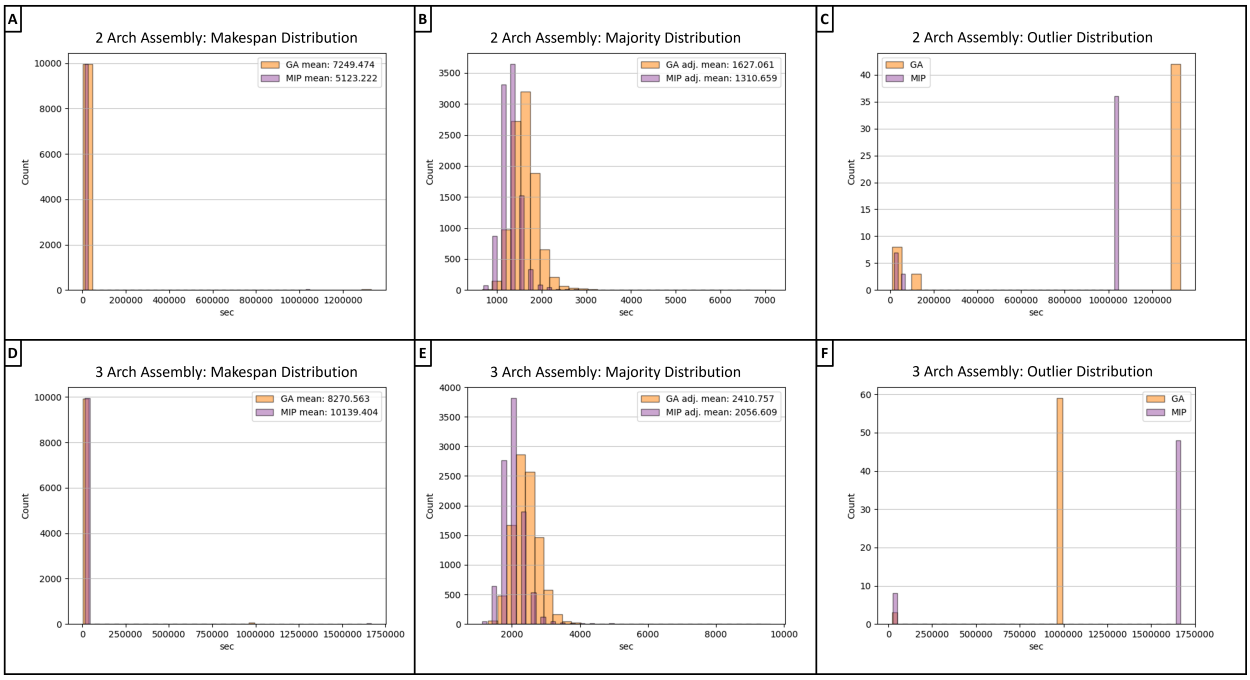


Figure C.2: All results using from 10,000 processing time samples. Panel A: Double arch assembly problem full distributions. Panel B: Double arch assembly problem majority distributions. Panel C: Double arch assembly problem outlier distributions. Panel D: Triple arch assembly problem full distributions. Panel E: Triple arch assembly problem majority distributions. Panel F: Triple arch assembly problem outlier distributions.

C.2 Processing Time Distributions

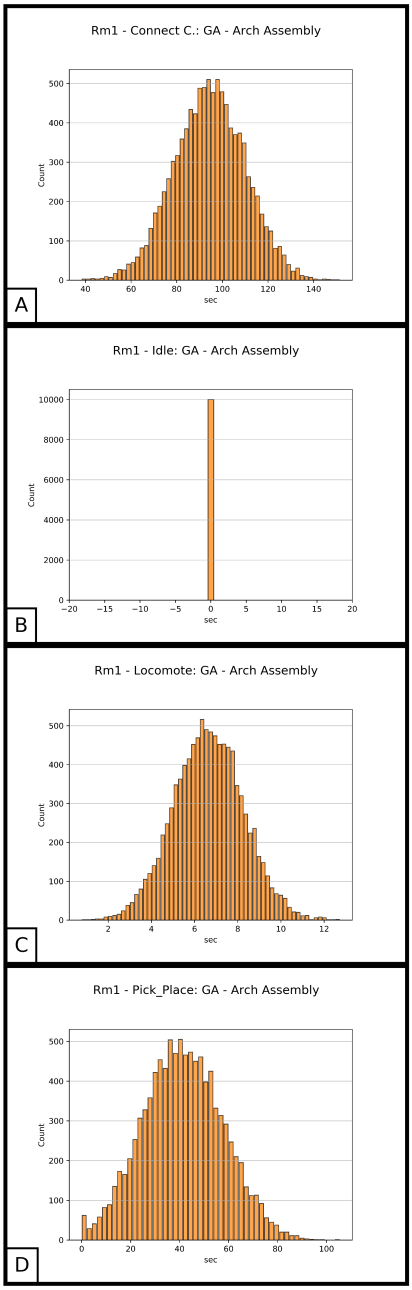


Figure C.3: 10,000 sample processing time distributions to build arch assembly according to GA schedule - Panel A: Robot 1 processing connect component operation. Panel B: Robot 1 processing idle operation. Panel C: Robot 1 processing locomote operation. Panel D: Robot 1 processing pick & place operation.

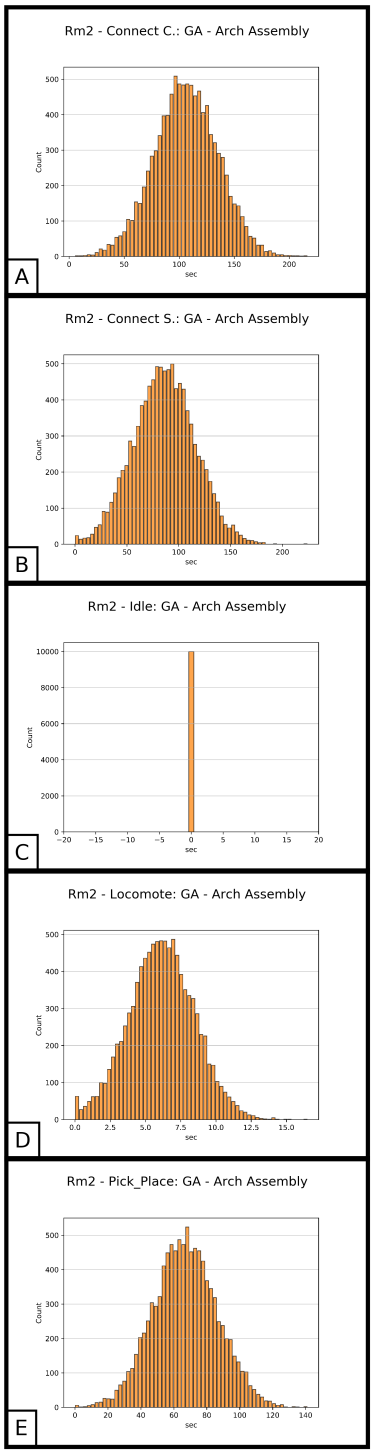


Figure C.4: 10,000 sample processing time distributions to build arch assembly according to GA schedule - Panel A: Robot 2 processing connect component operation. Panel B: Robot 2 processing connect subassembly operation. Panel C: Robot 2 processing idle operation. Panel D: Robot 2 processing locomote operation. Panel E: Robot 2 processing pick & place operation.

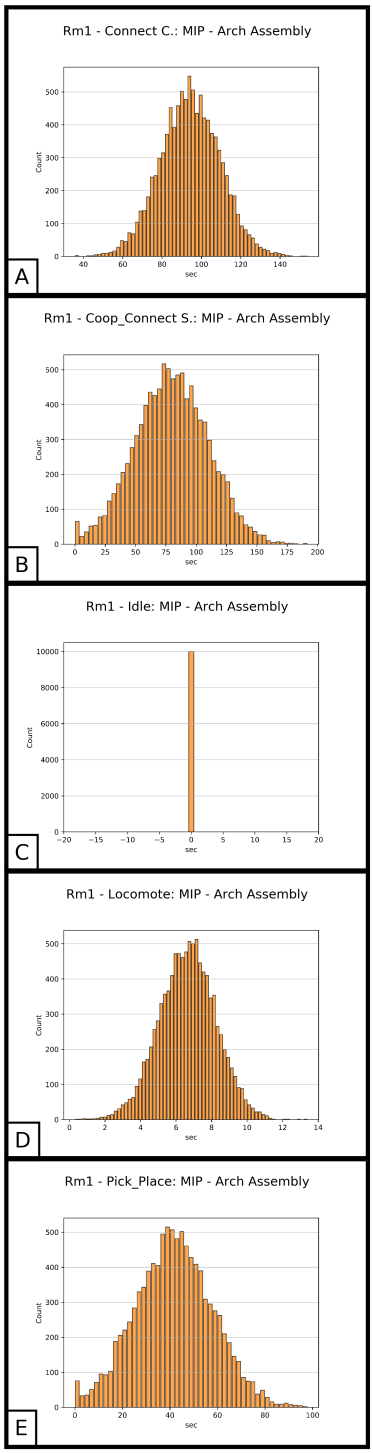


Figure C.5: 10,000 sample processing time distributions to build arch assembly according to MIP schedule - Panel A: Robot 1 processing connect component operation. Panel B: Robot 1 processing coop-connect subassembly operation. Panel C: Robot 1 processing idle operation. Panel D: Robot 1 processing locomote operation. Panel E: Robot 1 processing pick & place operation

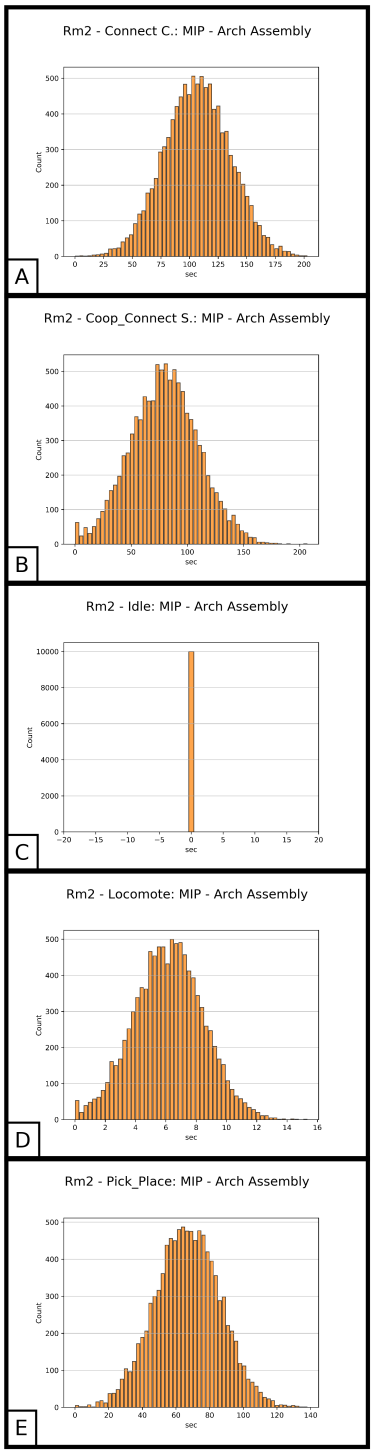


Figure C.6: 10,000 sample processing time distributions to build arch assembly according to MIP schedule - Panel A: Robot 2 processing connect component operation. Panel B: Robot 2 processing coop-connect subassembly operation. Panel C: Robot 2 processing idle operation. Panel D: Robot 2 processing locomote operation. Panel E: Robot 2 processing pick & place operation

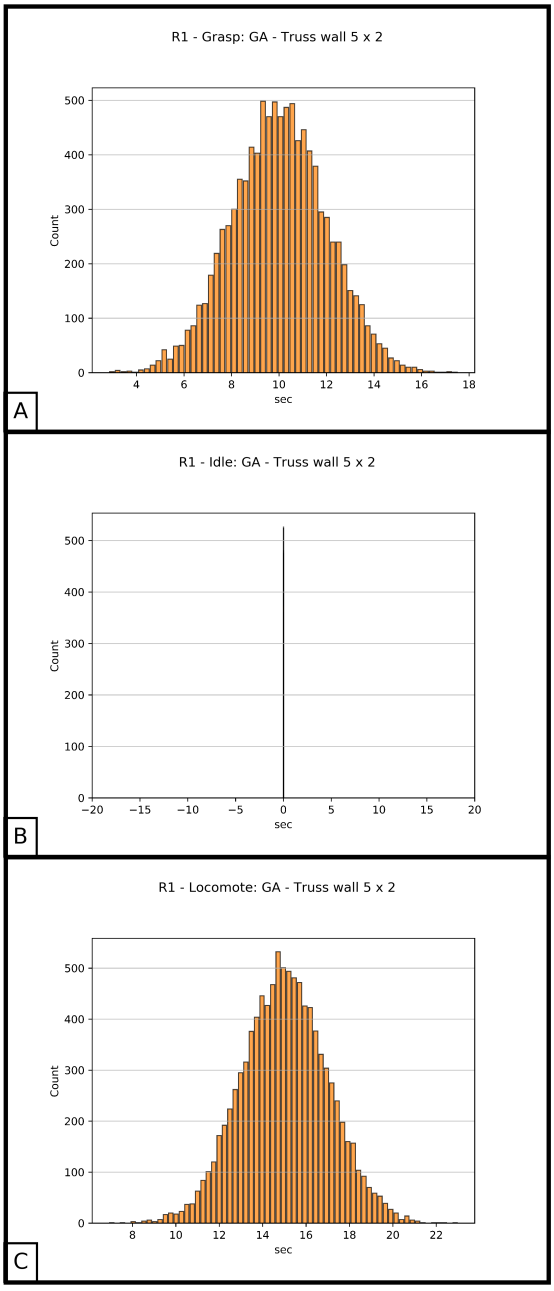


Figure C.7: 10,000 sample processing time distributions to build the 5x2 truss wall according to GA schedule - Panel A: Robot 1 processing grasp operation. Panel B: Robot 1 processing idle operation. Panel C: Robot 1 processing locomote operation.

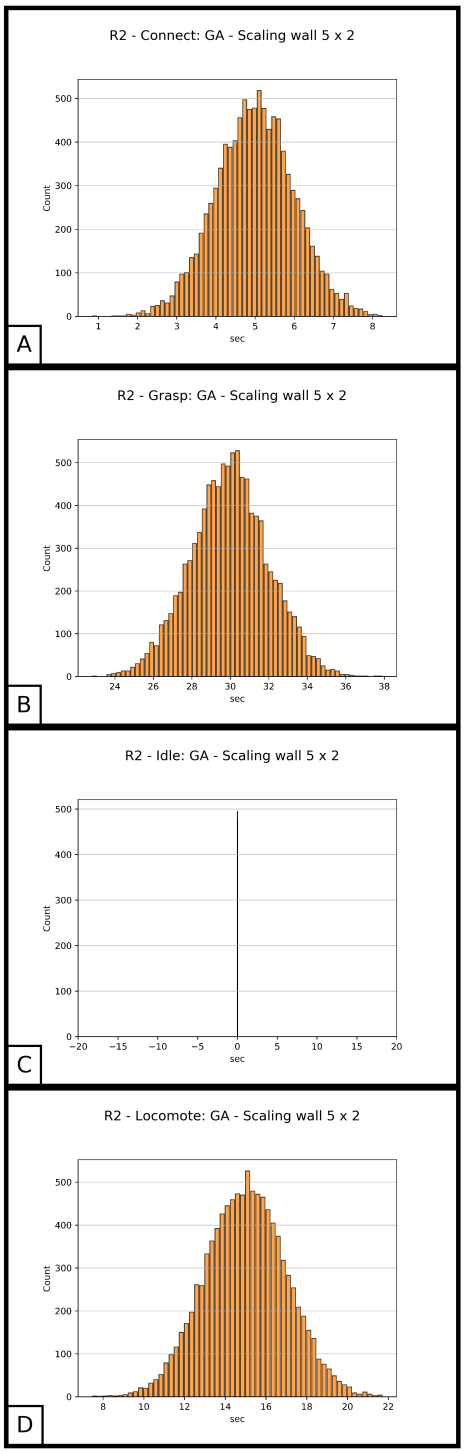


Figure C.8: 10,000 sample processing time distributions to build the 5x2 truss wall according to GA schedule - Panel A: Robot 2 processing connect operation. Panel B: Robot 2 processing grasp operation. Panel C: Robot 2 processing idle operation. Panel D: Robot 2 processing locomote operation.

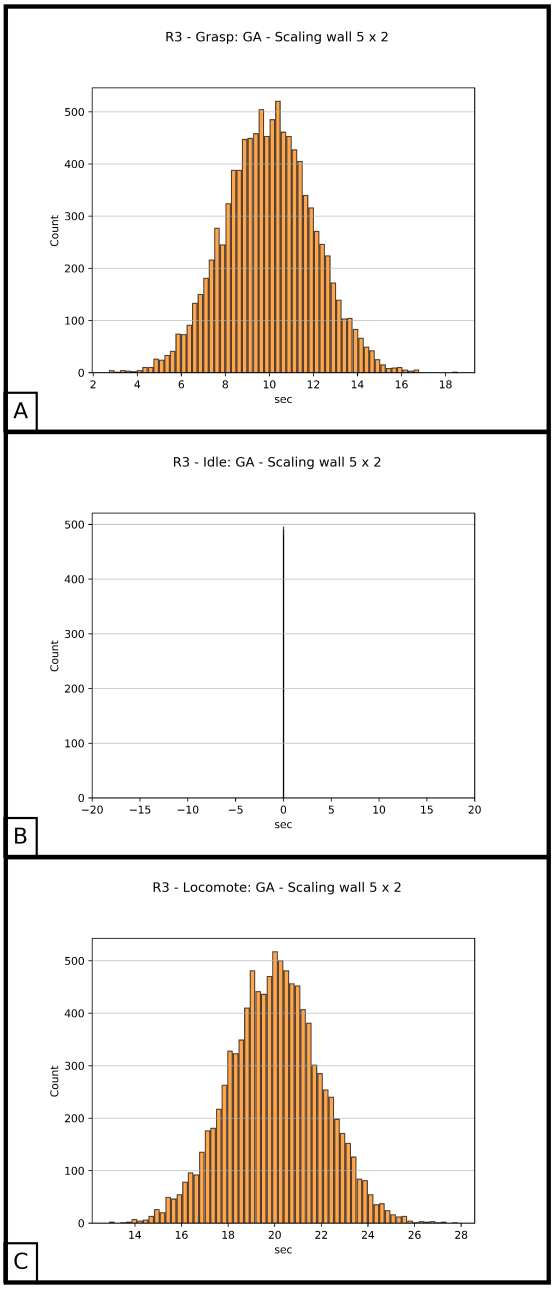


Figure C.9: 10,000 sample processing time distributions to build the 5x2 truss wall according to GA schedule - Panel A: Robot 3 processing grasp operation. Panel B: Robot 3 processing idle operation. Panel C: Robot 3 processing locomote operation.

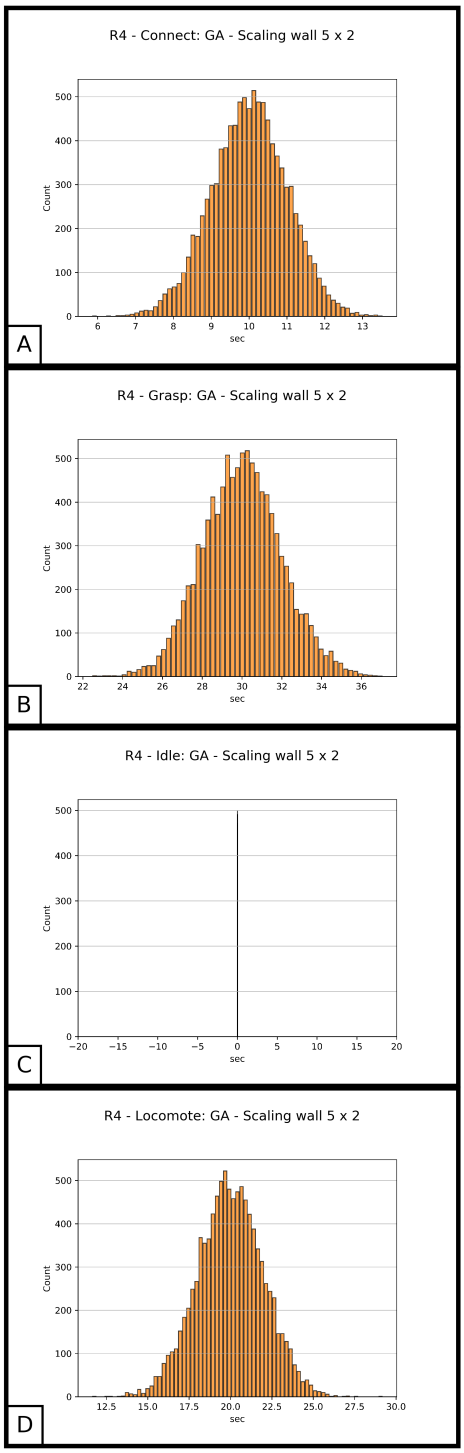


Figure C.10: 10,000 sample processing time distributions to build the 5x2 truss wall according to GA schedule - Panel A: Robot 4 processing connect operation. Panel B: Robot 4 processing grasp operation. Panel C: Robot 4 processing idle operation. Panel D: Robot 4 processing locomote operation.

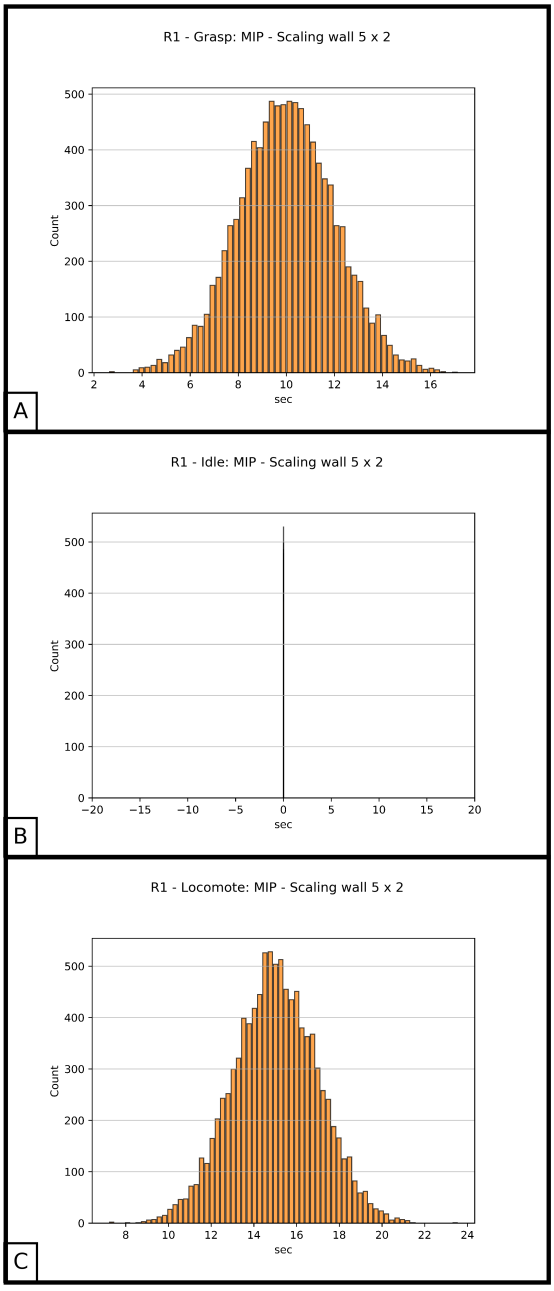


Figure C.11: 10,000 sample processing time distributions to build the 5x2 truss wall according to MIP schedule - Panel A: Robot 1 processing grasp operation. Panel B: Robot 1 processing idle operation. Panel C: Robot 1 processing locomote operation.

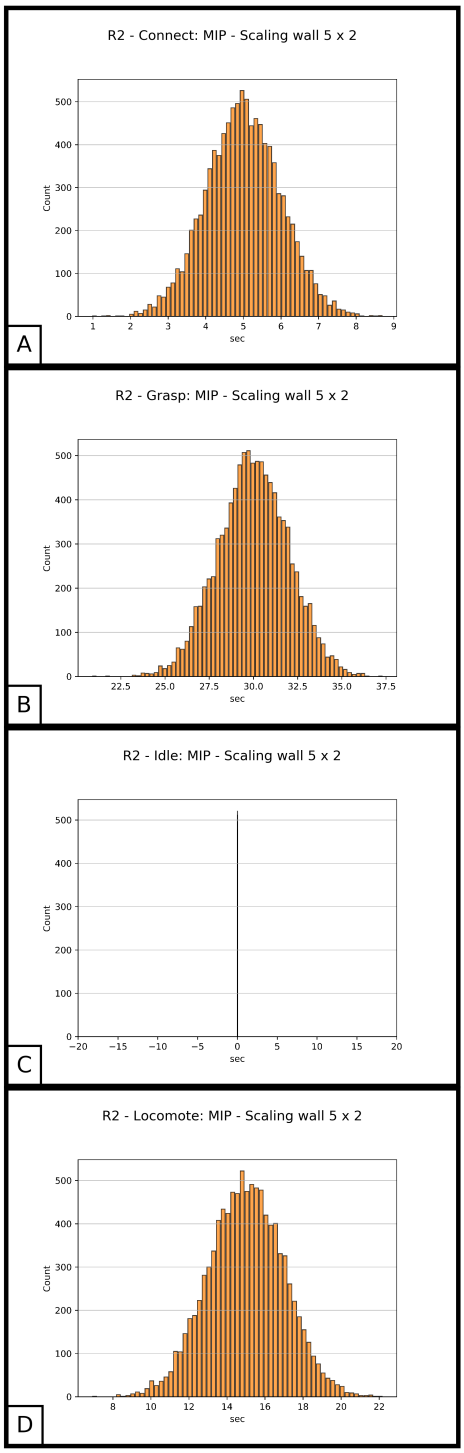


Figure C.12: 10,000 sample processing time distributions to build the 5x2 truss wall according to MIP schedule - Panel A: Robot 2 processing connect operation. Panel B: Robot 2 processing grasp operation. Panel C: Robot 2 processing idle operation. Panel D: Robot 2 processing locomote operation.

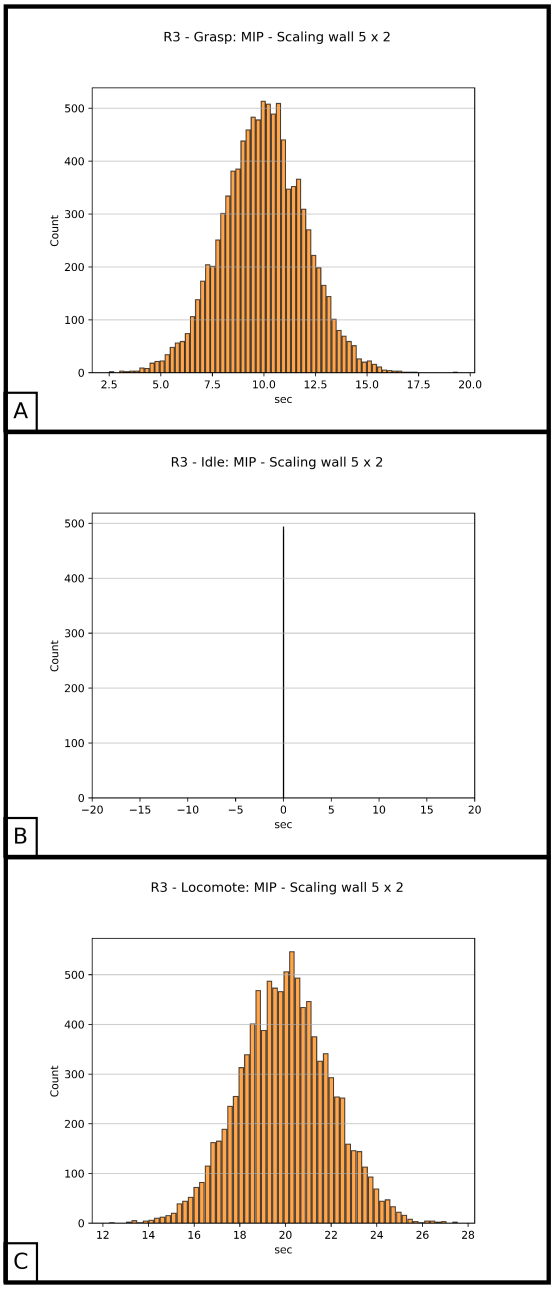


Figure C.13: 10,000 sample processing time distributions to build the 5x2 truss wall according to MIP schedule - Panel A: Robot 3 processing grasp operation. Panel B: Robot 3 processing idle operation. Panel C: Robot 3 processing locomote operation.

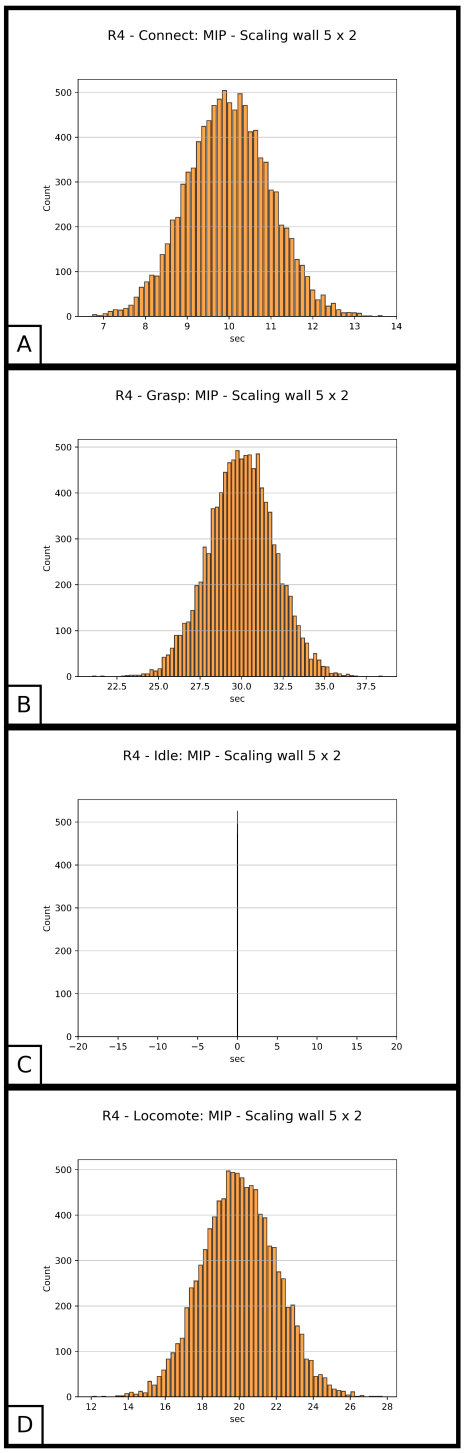


Figure C.14: 10,000 sample processing time distributions to build the 5x2 truss wall according to MIP schedule - Panel A: Robot 4 processing connect operation. Panel B: Robot 4 processing grasp operation. Panel C: Robot 4 processing idle operation. Panel D: Robot 4 processing locomote operation.

Appendix D

Ensemble Kalman Filter Algorithm

The Ensemble Kalman filter takes the form:

Algorithm 8 Ensemble Kalman Filter Algorithm

Require: $\hat{x}_{t-1|t-1}, u_t, z_t, N,$

1: $\hat{\overline{x}}_{t-1|t-1} = \frac{1}{N}\sum_{i=1}^{N} \hat{x}^i_{t-1|t-1}$

2: $\hat{y}^i_{t|t-1} = h(\hat{x}^i_{t-1|t-1}) + \mathcal{N}(O, R)$

3: $\hat{\overline{y}}_{t|t-1} = \frac{1}{N}\sum_{i=1}^{N} \hat{y}^i_{t|t-1}$

4: $P_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(\hat{x}^i_{t-1|t-1} - \hat{\overline{x}}_{t|t-1})(\hat{y}^i_{t|t-1} - \hat{\overline{y}}_{t|t-1})^T$

5: $P_{yy} = \frac{1}{N-1}\sum_{i=1}^{N}(\hat{y}^i_{t|t-1} - \hat{\overline{y}}_{t|t-1})(\hat{y}^i_{t|t-1} - \hat{\overline{y}}_{t|t-1})^T$

6: $K_t = P_{xy}P_{yy}^{-1}$

7: $\hat{x}^i_{t|t-1} = \hat{x}^i_{t-1|t-1} + K(z_t - \hat{y}^i_{t|t-1})$

8: $\hat{x}^i_{t|t} = f(\hat{x}^i_{t|t-1}, u_t) + \mathcal{N}(O, Q))$

9: $\hat{\overline{x}}_{t|t} = \frac{1}{N}\sum_{i=1}^{N} \hat{x}^i_{t|t}$

10: **return** $\hat{\overline{x}}_{t|t}, \hat{x}_{t|t}$

where N describes the number of samples generated by the filter to average in a Monte Carlo fashion. The first input is a concatenated vector of the particles produced in the previous time step, $\hat{x}_{t-1|t-1}$ The remaining inputs to the filter are the control commands given to the system at the current time step, u_t, the measurements taken at the current time step, z_t, and the number of particles, N generated by the filter to average in a Monte Carlo fashion.

The filter predicts a measurement, $\hat{y}^i_{t|t-1}$, based on each of the particles, factoring in the measurement noise, R. $\mathcal{N}(O, R)$ represents a multivariate normal distribution with zero mean and covariance R sampled for each particle. For the robotics application, this measurement

prediction step uses the forward kinematics to go from the joint values,θ, to the end effector location being measured.

The filter then finds a Kalman gain, K_t, based on the covariance of the state with the measurement, P_{xy}, and the measurement covariance, P_{yy}. This gain is used to weight the difference between the predicted measurement, $\hat{y}_{t|t-1}$, and the actual measurement, z_t, to update the particles, $\hat{x}_{t|t-1}$. These updated particles are then propagated using the control input, u_t and the noise the process noise, Q. $\mathcal{N}(O, Q)$ represents a multivariate normal distribution with zero mean and covariance Q sampled for each particle.

These propagated particles, $\hat{x}_{t|t}$, can then be averaged to provide the state estimate of the system, $\hat{\overline{x}}_{t|t}$. This estimate, along with the particles, are returned from the filter.

Appendix E

Ability Analysis Processing Time Converging Scatter Plots

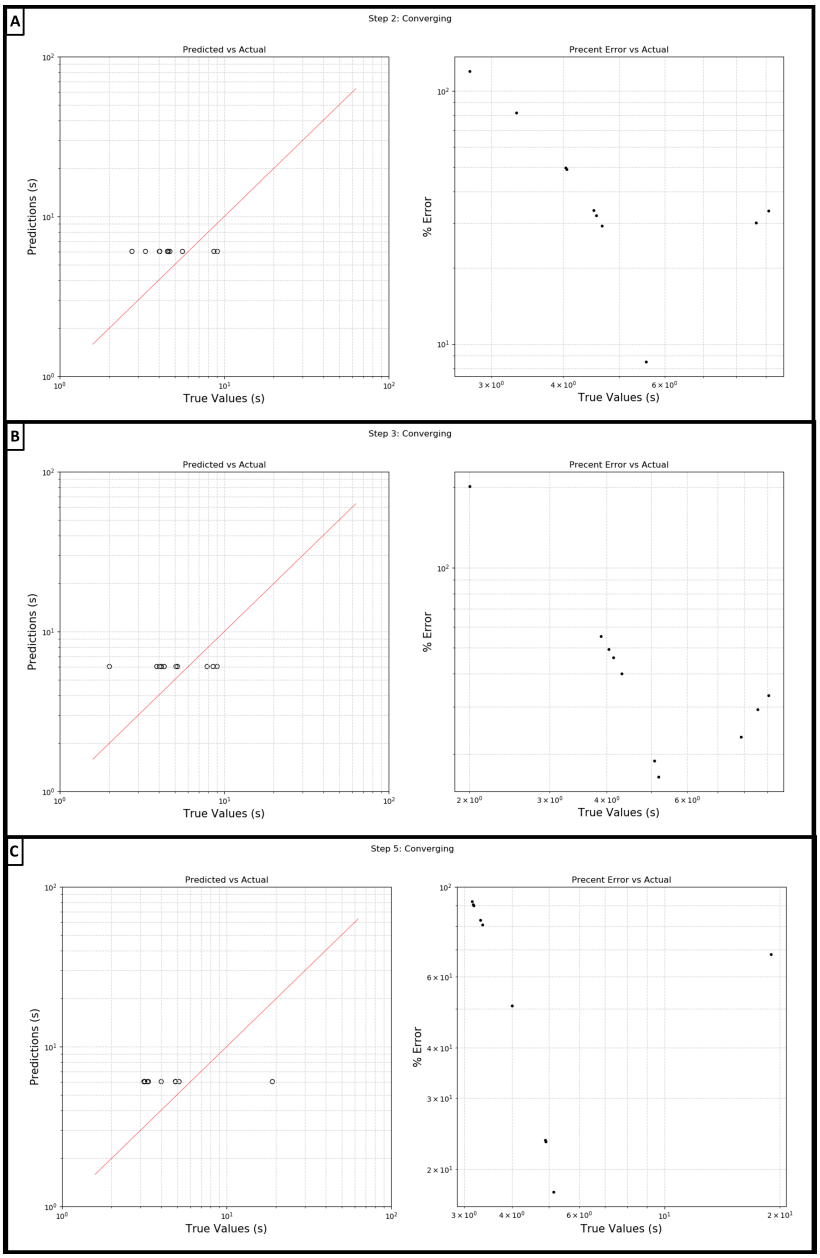


Figure E.1: Panel A: The log log scatter plot and log percent error of the convergence time present in step 2. Panel B: The log log scatter plot and log percent error of the convergence time present in step 3. Panel C: the log log scatter plot and log percent error of the convergence time present in step 5.